DB2.

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Common Criteria Certification: Administration and User Documentation – Revision 02

SC10-4313-00

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Common Criteria Certification: Administration and User Documentation – Revision 02

SC10-4313-00

Before using this information and the product it supports, be sure to read the general information under *Notices*.

Contents

Common Criteria certification of DB2 products

For Version 9.1, IBM DB2 products are certified according to the Common Criteria evaluation assurance level 4 (EAL4), augmented with Flaw remediation ALC_FLR.1. The following product is certified on the following operating systems:

Table 1. Certified configurations

	Windows Server 2003	**Red Hat Enterprise Linux 4**	**SuSE Linux Enterprise Server 9**	**AIX 5.3**	**Solaris 9**
IBM DB2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows	Yes	Yes	Yes	Yes	Yes

Notes:

1. For a Common Criteria certified DB2 environment, DB2 requires 64-bit Windows Server 2003 x64, Red Hat Enterprise Linux 4, or SuSE Linux Enterprise Server 9 operating systems for Intel EM64T- and AMD64-based systems.
2. In a Common Criteria certified DB2 environment, DB2 clients are supported on the following operating systems:
   - Windows 2003
   - Red Hat Enterprise Linux 4
   - SuSE Linux Enterprise Server 9
   - AIX 5.3
   - Solaris 9

For more information about Common Criteria, see the Common Criteria web site at: http://niap.nist.gov/cc-scheme/.

For information about installing and configuring a DB2 system that conforms to the Common Criteria EAL4, see the following books:

- *Common Criteria Certification: Installing IBM DB2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows*
- *Common Criteria Certification: Administration and User Documentation*

These books are available in PDF format from the DB2 Information Management Library.

Supported interfaces for a Common Criteria evaluated configuration

The set of DB2 interfaces that are used in the Common Criteria evaluation of DB2 are as follows:

- The DB2 install program
- The command line processor
- DB2 commands
- DB2 application programming interfaces (APIs)
- SQL statements

You can use these interfaces when installing and configuring a Common Criteria compliant DB2 system.

Other interfaces that are provided by DB2, such as the Control Center or Command Editor were not used during the Common Criteria evaluation of DB2, **and must not be used in the Common Criteria evaluation configuration.**

NOT FENCED routines are not supported.

About this book

This book is intended for use by assessors validating that DB2 conforms to the Common Critera EAL4 specification augmented with Flaw remediation ALC_FLR.1. It is also intended for those who want to set up a DB2 environment that conforms to the characteristics of the evaluated environment.

This book describes:

- The DB2 process model
- The DB2 security model, and the facilities available to set up and maintain security
- How to set up the DB2 environment so that it conforms to the requirements of the Common Criteria EAL4 specification.
- How to audit activity in the environment.
- Background information that you should be familiar with before setting up the DB2 UDB environment.
- Security-related considerations that are applicable to users of the DB2 UDB environment, including the type of authorization that the administrator must give to a user before that user can work with DB2 utilities.
- DB2 commands and SQL statements.
- The security-related considerations for writing applications that interact with DB2.
- Security plug-ins. Note that only the default IBM-supplied operating-system based authentication and group plug-ins are supported in Common Criteria compliant environments.

Note: This book does not provide information on how to install DB2. For installation information, see the *Common Criteria Certification: Installing IBM DB2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows*.

Some topics in this book and the *Common Criteria Certification: Installing IBM DB2 Version 9.1 Enterprise Server Edition for Linux, UNIX, and Windows*. book may link to topics that are not in either of these books. Topics that are referenced outside of the Common Criteria certification documentation are for informational purposes only, and are not required for either installing or configuring a Common Criteria compliant environment.

Part 1. Overview of the DB2 environment

Chapter 1. DB2 architecture and process overview

General information about DB2® architecture and processes can help you understand detailed information provided for specific topics.

The following figure shows a general overview of the architecture and processes for IBM® DB2 Version 9.1.

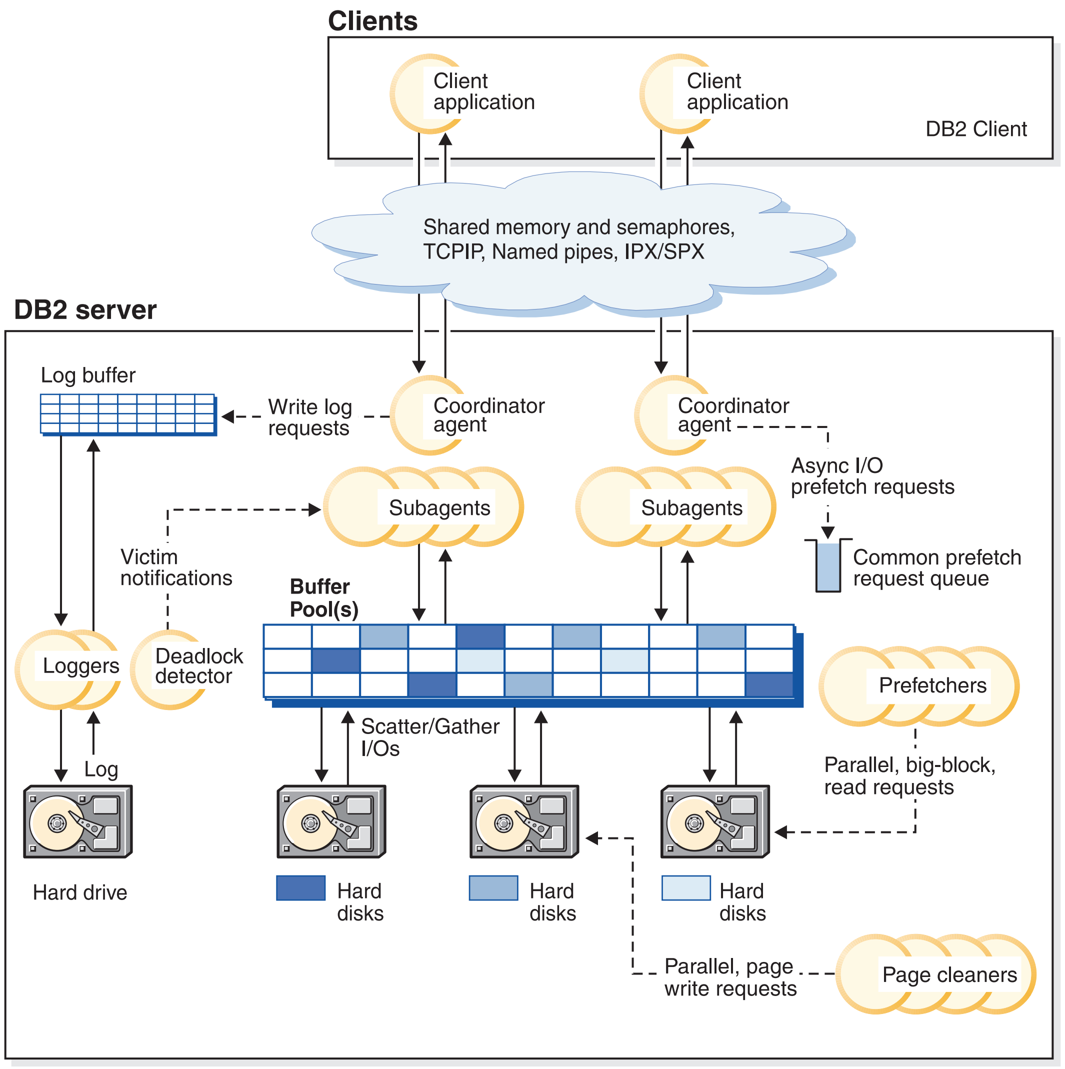


Figure 1. Architecture and Processes Overview

On the client side, either local or remote applications, or both, are linked with the DB2 client library. Local clients communicate using shared memory and semaphores; remote clients use a protocol such as Named Pipes (NPIPE), TCP/IP, NetBIOS, or SNA.

On the server side, activity is controlled by engine dispatchable units (EDUs). In all figures in this section, EDUs are shown as circles or groups of circles. EDUs are implemented as threads in a single process on Windows-based platforms and as processes on UNIX®. DB2 agents are the most common type of EDUs. These agents perform most of the SQL and XQuery processing on behalf of applications. Prefetchers and page cleaners are other common EDUs.

A set of subagents might be assigned to process the client application requests. Multiple subagents can be assigned if the machine where the server resides has multiple processors or is part of a partitioned database. For example, in a symmetric multiprocessing (SMP) environment, multiple SMP subagents can exploit the many processors.

All agents and subagents are managed using a pooling algorithm that minimizes the creation and destruction of EDUs.

Buffer pools are areas of database server memory where database pages of user table data, index data, and catalog data are temporarily moved and can be modified. Buffer pools are a key determinant of database performance because data can be accessed much faster from memory than from disk. If more of the data needed by applications is present in a buffer pool, less time is required to access the data than to find it on disk.

The configuration of the buffer pools, as well as prefetcher and page cleaner EDUs, controls how quickly data can be accessed and how readily available it is to applications.

- **Prefetchers** retrieve data from disk and move it into the buffer pool before applications need the data. For example, applications needing to scan through large volumes of data would have to wait for data to be moved from disk into the buffer pool if there were no data prefetchers. Agents of the application send asynchronous read-ahead requests to a common prefetch queue. As prefetchers become available, they implement those requests by using big-block or scatter-read input operations to bring the requested pages from disk to the buffer pool. If you have multiple disks for storage of the database data, the data can be striped across the disks. Striping data lets the prefetchers use multiple disks at the same time to retrieve data.
- **Page cleaners** move data from the buffer pool back out to disk. Page cleaners are background EDUs that are independent of the application agents. They look for pages from the buffer pool that are no longer needed and write the pages to disk. Page cleaners ensure that there is room in the buffer pool for the pages being retrieved by the prefetchers.

Without the independent prefetchers and the page cleaner EDUs, the application agents would have to do all of the reading and writing of data between the buffer pool and disk storage.

Related concepts:

- “Connection-concentrator improvements for client connections” in *Performance Guide*
- “Deadlocks” on page 357
- “Prefetching data into the buffer pool” in *Performance Guide*
- “Database directories and files” on page 239
- “Reducing logging overhead to improve query performance” in *Performance Guide*
- Chapter 3, “Client-server processing model,” on page 13
- “Update processing” in *Performance Guide*

Related reference:

- “max_connections - Maximum number of client connections configuration parameter” in *Performance Guide*
- “max_coordagents - Maximum number of coordinating agents configuration parameter” in *Performance Guide*

Chapter 2. The DB2 Process Model

Knowledge of the DB2 process model can help you determine the nature of a problem because it helps you to understand how the database manager and its associated components interact.

The process model used by all DB2 servers facilitates the communication that occurs between database servers and clients and local applications. It also ensures that database applications are isolated from resources such as database control blocks and critical database files.

UNIX-based environments use an architecture based on **system processes**. For example, the DB2 communications listeners are created as system processes. Intel™ operating systems such as Windows® use an architecture based on **threads** to maximize performance. Unless explicitly noted, this discussion uses the term ″process″ to refer to both processes and threads. You can find details of the differences between the use of Windows threads and UNIX processes later in this topic.

For each database being accessed, various processes are started to deal with the various database tasks such as prefetching, communication, and logging.

Each process of a client application has a single **coordinator agent** that operates on a database. A coordinator agent works on behalf of an application, and communicates to other agents, using interprocess communication (IPC) or remote communication protocols.

DB2 architecture provides a **firewall** so that applications run in a different address space from DB2. The firewall protects the database and the database manager from applications, stored procedures, and user-defined functions (UDFs). A firewall maintains the integrity of the data in the databases, because it disables application programming errors from overwriting internal buffers or files of the database manager. The firewall also improves reliability, because application errors cannot crash the database manager.

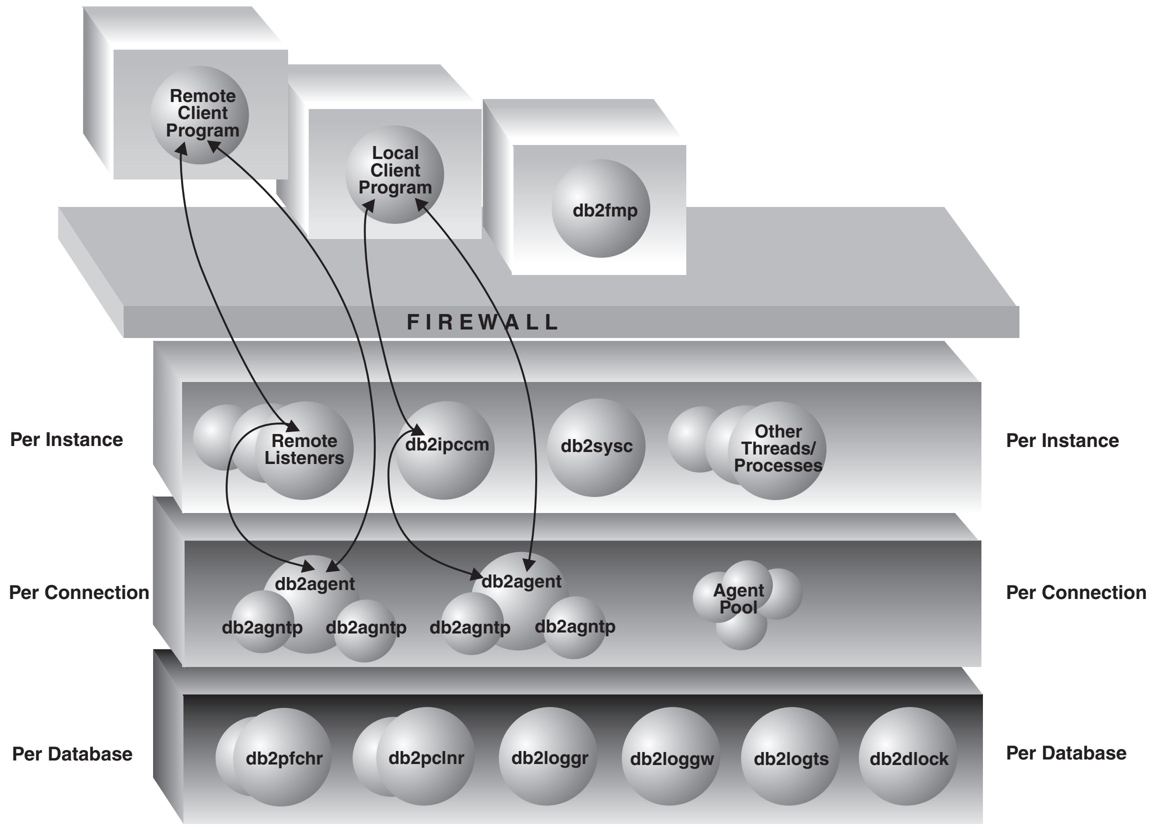


Figure 2. Process Model for DB2 Systems

The following list provides additional details on the processes shown in the figure:

Client Programs:

Client programs run remotely or on the same machine as the database server. They make their first contact with the database through a listener. A coordinator agent (db2agent) is then assigned to them.

Listeners:

Client programs make initial contact with communication listeners, which are started when DB2 is started. There is a listener for each configured communication protocol, and an interprocess communications (IPC) listener (db2ipccm) for local client programs. Listeners include:

- **db2ipccm**, for local client connections
- **db2tcpcm**, for TCP/IP connections
- **db2snacm**, for APPC connections
- **db2tcpdm**, for TCP/IP discovery tool requests

Agents:

All connection requests from client applications, whether they are local or remote, are allocated a corresponding coordinator agent (**db2agent**). When the coordinator agent is created, it performs all database requests on behalf of the application.

In some environments, in which the *intra_parallel* database manager configuration parameter is enabled, the coordinator agent distributes the database requests to subagents (`db2agntp`). These agents perform the requests for the application. Once the coordinator agent is created, it handles all database requests on behalf of its application by coordinating subagents (`db2agntp`) that perform requests on the database. Subagents that are associated with an application but are currently idle are identified by the process name **db2agnta**. Independent coordinator agents are identified by the process name **db2agnti**.

A coordinator agent may be:
- Connected to the database with an alias. For example, ″db2agent (DATA1)″ is connected to the database alias ″DATA1″.
- Attached to an instance. For example, ″db2agent (user1)″ is attached to the instance ″user1″.

An additional type of agent, **db2agnsc** is created by DB2 to perform certain operations in parallel. In particular, they are used in database recovery actions.

Idle agents reside in an agent pool. These agents are available for requests from coordinator agents operating on behalf of client programs, or from subagents operating on behalf of existing coordinator agents. The number of available agents is dependent on the database manager configuration parameters *maxagents* and *num_poolagents*.

db2fmp:

The fenced mode process. It is responsible for executing fenced stored procedures and user-defined functions outside the firewall. db2fmp is always a separate process but may be multithreaded depending on the types of routines it executes.

Database Threads/Processes:

The following list includes some of the important threads/processes used by each database:
- **db2pfchr**, for buffer pool prefetchers
- **db2pclnr**, for buffer pool page cleaners
- **db2loggr**, for manipulating log files to handle transaction processing and recovery
- **db2loggw**, for writing log records to the log files.
- **db2logts**, for collecting historical information about which logs are active when a tablespace is modified. This information is ultimately recorded in the DB2TSCHG.HIS file in the database directory. It is used to speed up tablespace rollforward recovery.
- **db2dlock**, for deadlock detection. In a multi-partitioned database environment, an additional process called **db2glock** is used to coordinate the information gathered from the db2dlock process on each partition. db2glock runs only on the catalog partition.
- **db2taskd**, for distribution of background database tasks. The tasks are executed by processes called **db2taskp**.
- **db2hadrp**, HADR primary server process
- **db2hadrs**, HADR standby server process
- **db2lfr**, for log file readers that processes individual log files
- **db2logts** tracks which table spaces have log records in each log file

- **db2shred** processes individual log records within log pages
- **db2redom**, for the redo master. During recovery, processes redo log records and assigns log records to redo workers for processing.
- **db2redow**, for the redo worker. During recovery, processes redo log records at the request of the redo master.
- **db2logmgr**, for the log manager. Manages log files for a recoverable database
- event monitor processes are identified as follows:
  - **db2evm%1%2 (%3)** where **%1** can be
    - **g**- global file event monitor
    - **l** - local file event monitor
    - **t** - table event monitor
    - **gp** - global piped event monitor
    - **lp** - local piped event monitor

    where **%2** can be
    - **i** - coordinator
    - **p** - not coordinator

    and **%3** is the event monitor name
- backup and restore processes are identified as follows:
  - **db2bm.%1.%2** backup and restore buffer manipulator, and **db2med.%1.%2**, backup and restore media controller, where%
    - **%1**- the process number of the agent that controls the backup or restore session
    - **%2**- a sequential value used to disambiguate among (possibly many) processes belonging to a particular backup or restore session

    For example: `db2bm.13579.2` identifies the second db2bm process that is controlled by the db2agent process with pid 13579.

Database Server Threads and Processes:

The system controller (**db2sysc**) must exist in order for the database server to function. Also, the following threads and processes may be started to carry out various tasks:

- **db2resync**, the resync agent that scans the global resync list
- **db2gds**, the global daemon spawner on UNIX-based systems that starts new processes
- **db2wdog**, the watchdog on UNIX-based systems that handles abnormal terminations
- **db2fcms**, the fast communications manager sender daemon
- **db2fcmr**, the fast communications manager receiver daemon
- **db2pdbc**, the parallel system controller, which handles parallel requests from remote nodes (used only in a partitioned database environment).
- **db2cart**, for archiving log files when accessing a database configured with USEREXIT enabled
- **db2fmtlg**, for formatting log files, when accessing a database configured with LOGRETAIN enabled, but with USEREXIT disabled
- **db2panic**, the panic agent, which handles urgent requests after agent limits have been reached at a particular node (used only in a partitioned database environment)

- **db2srvlst**, manages lists of addresses for systems such as DB2 for z/OS.
- **db2fmd**, the fault monitor daemon
- **db2disp**, the client connection concentrator dispatcher
- **db2acd**, autonomic computing daemon hosting the health monitor and automatic maintenance utilities. This process was formerly called **db2hmon**.

Differences between Windows and UNIX

DB2 for Windows differs from UNIX-based environments in that the database engine is multi-threaded, not multi-processed. In Windows systems, each of the dispatchable units on the agent side of the firewall is a thread under the process db2sysc. This allows the database engine to let the operating system perform task-switching at the thread level instead of the process level. For each database being accessed, threads are started to deal with database tasks (for example, prefetching).

Another difference is in the handling of abnormal terminations. There is no need for a ″watchdog″ process in Windows systems, because these systems ensure that the allocated resources are cleaned up after an abnormal termination. Thus, there is no equivalent of the db2wdog process on the Windows systems. In addition, the db2gds process or thread is not needed on the Windows systems, which have their own mechanisms for starting threads.

Related concepts:

- “First failure data capture information” in *Troubleshooting Guide*
- “Getting started with the Replication Center” in *Troubleshooting Guide*

Chapter 3. Client-server processing model

Local and remote application processes can work with the same database. A remote application is one that initiates a database action from a machine that is remote from the database machine. Local applications are directly attached to the database at the server machine.

How DB2 manages client connections depends on whether the connection concentrator is on or off. The connection concentrator is ON when the *max_connections* database manager configuration parameter is set larger than the *max_coordagents* configuration parameter.

- If the connection concentrator is OFF, each client application is assigned a unique EDU called a *coordinator agent* that coordinates the processing for that application and communicates with it.
- If the connection concentrator is ON, each coordinator agent can manage many client connections, one at a time, and might coordinate the other worker agents to do this work. For Internet applications with many relatively transient connections, or similar applications with many relatively small transactions, the connection concentrator improves performance by allowing many more client applications to be connected. It also reduces system resource use for each connection.

Each of the circles of the following figure represent engine dispatchable units (EDUs) which are known as “processes” on UNIX platforms, and “threads” on Windows Operating systems.

A means of communicating between an application and the database manager must be established before the work the application wants done at the database can be carried out.

At A1 in the figure below, a local client establishes communications first through the db2ipccm. At A2, the db2ipccm works with a db2agent EDU, which becomes the coordinator agent for the application requests from the local client. The coordinator agent then contacts the client application at A3 to establish shared memory communications between the client application and the coordinator. The application at the local client is connected to the database at A4.

At B1 in the figure below, a remote client establishes communications through the db2tcpcm EDU. If any other communications protocol is chosen, the appropriate communication manager is used. The db2tcpcm EDU establishes TCP/IP communication between the client application and the db2tcpcm. It then works with a db2agent at B2, which becomes the coordinator agent for the application and passes the connection to this agent. At B3 the coordinator agent contacts the remote client application and is connected to the database.

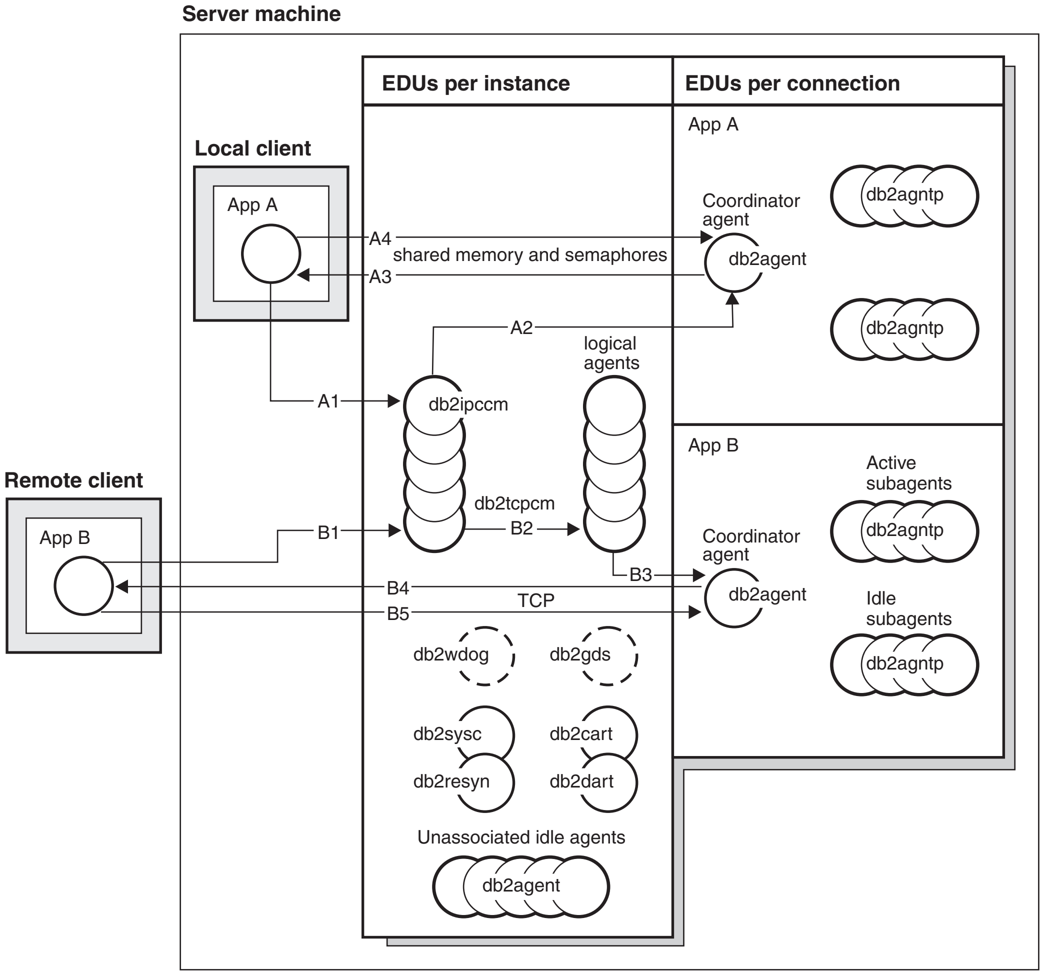


Figure 3. Process model overview

Other things to notice in this figure:

- Worker agents carry out application requests.
- There are four types of worker agents: active coordinator agents, active subagents, associated subagents, and idle agents.
- Each client connection is linked to an active coordinator agent.
- In a partitioned database environment, and enabled intra-partition parallelism environments, the coordinator agents distribute database requests to subagents (db2agntp). The subagents perform the requests for the application.
- There is an agent pool (db2agent) where idle and pooled agents wait for new work.
- Other EDUs manage client connections, logs, two-phase COMMITs, backup and restore tasks, and other tasks.

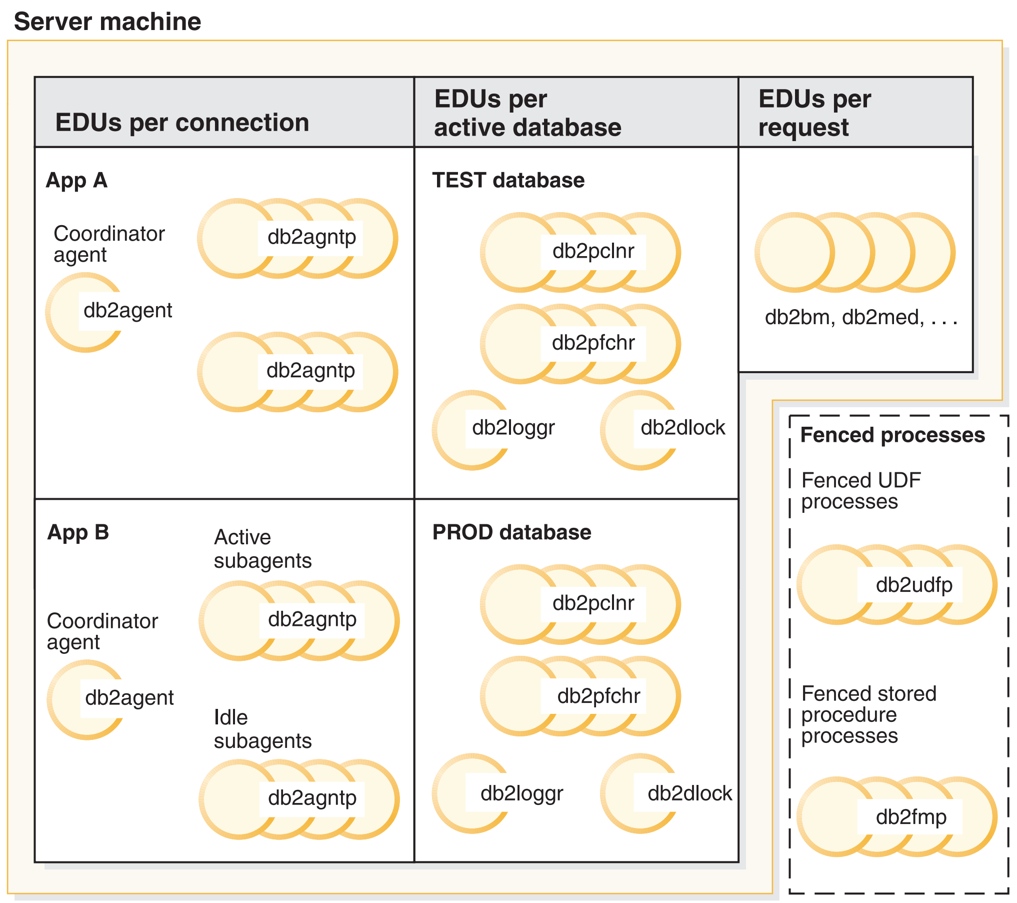


Figure 4. Process model, part 2

This figure shows additional engine dispatchable units (EDUs) that are part of the server machine environment. Each active database has its own shared pool of prefetchers (db2pfchr) and page cleaners (db2pclnr), and its own logger (db2loggr) and deadlock detector (db2dlock).

Fenced user-defined functions (UDFs) and stored procedures, which are not shown in the figure, are managed to minimize costs associated with their creation and destruction. The default for the *keepfenced* database manager configuration parameter is "YES", which keeps the stored procedure process available for re-use at the next stored procedure call.

Note: Unfenced UDFs and stored procedures run directly in an agent's address space for better performance. However, because they have unrestricted access to the agent's address space, they need to be rigorously tested before being used.

The multiple database partition processing model is a logical extension of the single database partition processing model. In fact, a single common code base supports both modes of operation. The following figure shows the similarities and differences between the single database partition processing model as seen in the previous two figures, and the multiple database partition processing model.

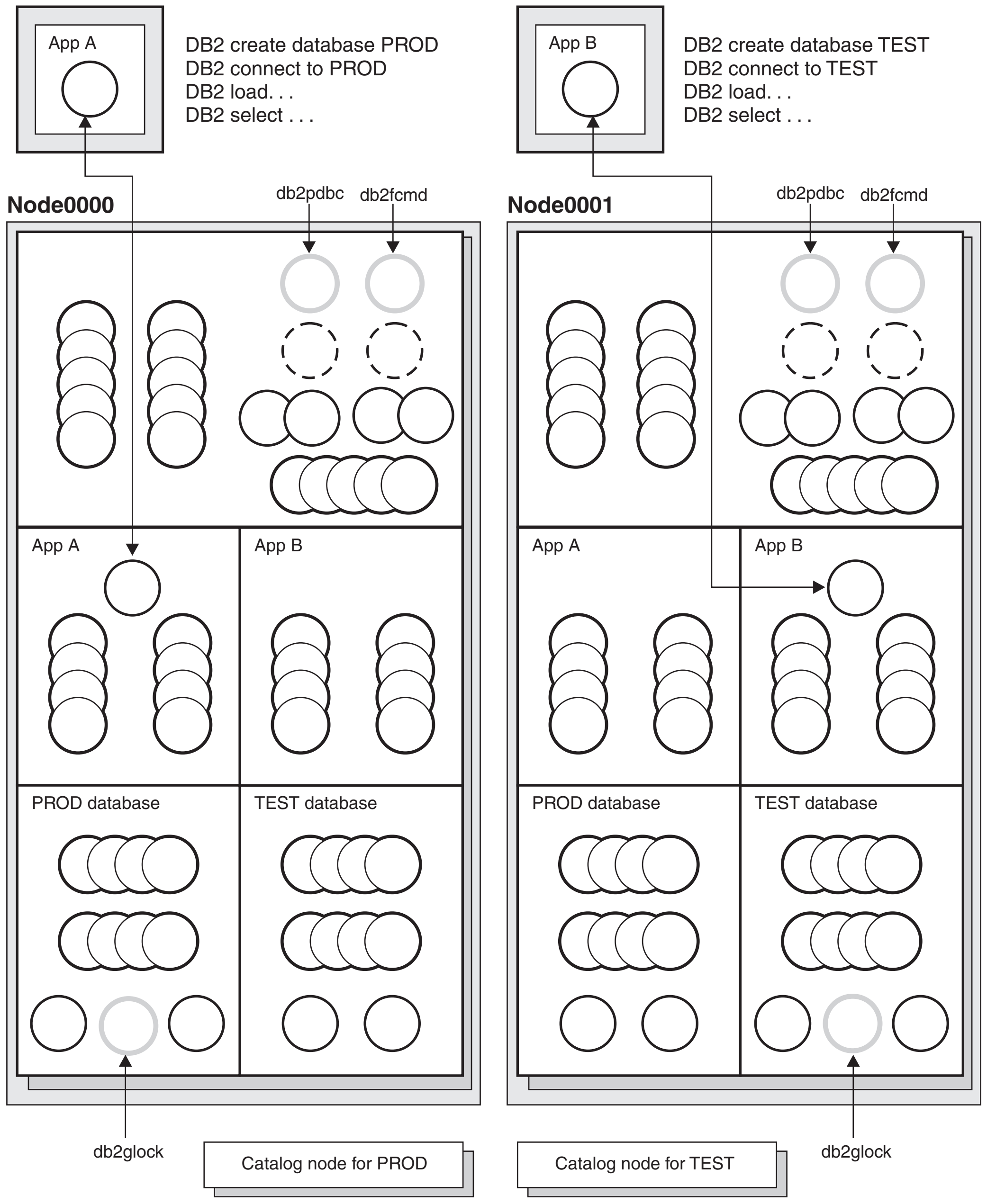


Figure 5. Process model and multiple database partitions

Most engine dispatchable units (EDUs) are the same between the single database partition processing model and the multiple database partition processing model.

In a multiple database partition (or node) environment, one of the database partitions is the catalog node. The catalog keeps all of the information relating to the objects in the database.

As shown in the figure above, because Application A creates the PROD database on Node0000, the catalog for the PROD database is created on this node. Similarly, because Application B creates the TEST database on Node0001, the catalog for the TEST database is created on this node. You might want to create your databases on different nodes to balance the extra activity associated with the catalogs for each database across the nodes in your system environment.

There are additional EDUs (db2pdbc and db2fcmd) associated with the instance and these are found on each node in a multiple partition database environment. These EDUs are needed to coordinate requests across database partitions and to enable the Fast Communication Manager (FCM).

There is also an additional EDU (db2glock) associated with the catalog node for the database. This EDU controls global deadlocks across the nodes where the active database is located.

Each CONNECT from an application is represented by a connection that is associated with a coordinator agent to handle the connection. The *coordinator agent* is the agent that communicates with the application, receiving requests and sending replies. It can either satisfy the request itself or coordinate multiple subagents to work on the request. The database partition where the coordinator agent exists is called the *coordinator node* of that application. The coordinator node can also be set with the SET CLIENT CONNECT_NODE command.

Parts of the database requests from the application are sent by the coordinator node to subagents at the other database partitions; and all results from the other database partitions are consolidated at the coordinator node before being sent back to the application.

The database partition where the CREATE DATABASE command was issued is called the "catalog node" for the database. It is at this database partition that the catalog tables are stored. Typically, all user tables are distributed across a set of nodes.

Note: Any number of database partitions can be configured to run on the same machine. This is known as a "multiple logical partition", or "multiple logical node", configuration. Such a configuration is very useful on large symmetric multiprocessor (SMP) machines with very large main memory. In this environment, communications between database partitions can be optimized to use shared memory and semaphores.

Related concepts:

- Chapter 1, "DB2 architecture and process overview," on page 3
- "Connection-concentrator improvements for client connections" in *Performance Guide*
- "Reducing logging overhead to improve query performance" in *Performance Guide*
- "Update processing" in *Performance Guide*

Chapter 4. Database agents

For each database that an application accesses, various processes or threads start to perform the various application tasks. These tasks include logging, communication, and prefetching.

Database agents are engine dispatchable unit (EDU) processes or threads. Database agents do the work in the database manager that applications request. In UNIX environments, these agents run as processes. In Intel-based operating systems such Windows, the agents run as threads.

The maximum number of application connections is controlled by the *max_connections* database manager configuration parameter. The work of each application connection is coordinated by a single worker agent.

A *worker agent* carries out application requests but has no permanent attachment to any particular application. The coordinator worker agent has all the information and control blocks required to complete actions within the database manager that were requested by the application.

There are four types of worker agents:
- Idle agents
- Inactive agents
- Active coordinator agents
- Subagents

Idle agents
This is the simplest form of worker agent. It does not have an outbound connection and it does not have a local database connection or an instance attachment.

Inactive agents
An inactive agent is a worker agent that is not in an active transaction, does not have an outbound connection, and does not have a local database connection or an instance attachment. Inactive agents are free to begin doing work for an application connection.

Active coordinator agents
Each process or thread of a client application has a single active agent that coordinates its work on a database. After the coordinator agent is created, it performs all database requests on behalf of its application, and communicates to other agents using inter-process communication (IPC) or remote communication protocols. Each agent operates with its own private memory and shares database manager and database global resources such as the buffer pool with other agents. When a transaction completes, the active coordinator agent may become an inactive agent.

When a client disconnects from a database or detaches from an instance its coordinating agent will be:
- An active agent. If other connections are waiting, the worker agent becomes an active coordinator agent.
- Freed and marked as idle, if no connections are waiting and the maximum number of pool agents has not been reached.

- Terminated and its storage freed, if no connections are waiting and the maximum number of pool agents has been reached.

Subagents
: In partitioned database environments and environments with intra-partition parallelism enabled, the coordinator agent distributes database requests to subagents, and these agents perform the requests for the application. After the coordinator agent is created, it handles all database requests on behalf of its application by coordinating the subagents that perform requests on the database.

Agents that are not performing work for any applications and that are waiting to be assigned are considered to be idle agents and reside in an *agent pool*. These agents are available for requests from coordinator agents operating for client programs or for subagents operating for existing coordinator agents. The number of available agents depends on the database manager configuration parameters *maxagents* and *num_poolagents*.

When an agent finishes its work but still has a connection to a database, it is placed in the agent pool. Regardless of whether the connection concentrator is enabled for the database, if an agent is not waked up to serve a new request within a certain period of time and the current number of active and pooled agents is greater than *num_poolagents*, the agent is terminated.

Agents from the agent pool (*num_poolagents*) are re-used as coordinator agents for the following kinds of applications:
- Remote TCP/IP-based applications
- Local applications on UNIX-based operating systems
- Both local and remote applications on Windows operating systems.

Other kinds of remote applications always create a new agent. If no idle agents exist when an agent is required, a new agent is created dynamically. Because creating a new agent requires a certain amount of overhead CONNECT and ATTACH performance is better if an idle agent can be activated for a client.

When a subagent is performing work for of an application, it is *associated* with that application. After it completes the assigned work, it can be placed in the agent pool, but it remains associated with the original application. When the application requests additional work, the database manager first checks the idle pool for associated agents before it creates a new agent.

Related concepts:
- "Agents in a partitioned database" in *Performance Guide*
- "Configuration parameters that affect the number of agents" on page 55
- "Connection-concentrator improvements for client connections" in *Performance Guide*
- Chapter 5, "Database agent management," on page 21

Chapter 5. Database agent management

Most applications establish a one-to-one relationship between the number of connected applications and the number of application requests that can be processed by the database. However, it may be that your work environment is such that you require a many-to-one relationship between the number of connected applications and the number of application requests that can be processed.

The ability to control these factors separately is provided by two database manager configuration parameters:

- The *max_connections* parameter, which specifies the number of connected applications
- The *max_coordagents* parameter, which specifies the number of application requests that can be processed

The connection concentrator is enabled when the value of *max_connections* is greater than the value of *max_coordagents*.

Because each active coordinator agents requires global resource overhead, the greater the number of these agents the greater the chance that the upper limits of available database global resources will be reached. To prevent reaching the upper limits of available database global resources, you might set the value of *max_connections* higher than the value of *max_coordagents*.

Related concepts:

- "Agents in a partitioned database" in *Performance Guide*
- "Configuration parameters that affect the number of agents" on page 55
- "Connection-concentrator improvements for client connections" in *Performance Guide*

Chapter 6. Naming rules

General naming rules

Rules exist for the naming of all objects, users and groups. Some of these rules are specific to the platform you are working on. For example, there is a rule regarding the use of upper and lowercase letters in a name.

- On UNIX platforms, names must be in lowercase.
- On Windows platforms, names can be in upper, lower, and mixed-case.

Unless otherwise specified, all names can include the following characters:

- A through Z. When used in most names, characters A through Z are converted from lowercase to uppercase.
- 0 through 9.
- ! % ( ) { } . – ^ ~ _ (underscore) @, #, $, and space.
- \ (backslash).

Names cannot begin with a number or with the underscore character.

Do not use SQL reserved words to name tables, views, columns, indexes, or authorization IDs.

There are other special characters that might work separately depending on your operating system and where you are working with the DB2 database. However, while they might work, there is no guarantee that they will work. It is not recommended that you use these other special characters when naming objects in your database.

User and group names also need to follow the rules forced on specific operation systems by the related systems. For example, on Linux and UNIX platforms, user names and primary group names must follow these rules:

- Allowed characters: lowercase a through z, 0 through 9, and _ (underscore) for names not starting with 0 through 9.
- Length must be less than or equal to 8 characters.

You also need to consider object naming rules, workstation naming rules, naming rules in an NLS environment, and naming rules in a Unicode environment.

Related concepts:

- "DB2 database object naming rules" on page 24
- "Federated database object naming rules" in *Administration Guide: Implementation*
- "User, user ID and group naming rules" on page 26
- "Workstation naming rules" on page 26

DB2 database object naming rules

All objects follow the General Naming Rules. In addition, some objects have additional restrictions shown in the accompanying tables.

Table 2. Database, database alias and instance naming rules

Objects	Guidelines
• Databases • Database aliases • Instances	• Database names must be unique within the location in which they are cataloged. On Linux™ and UNIX implementations of the DB2 database manager, this location is a directory path, while on Windows implementations, it is a logical disk. • Database alias names must be unique within the system database directory. When a new database is created, the alias defaults to the database name. As a result, you cannot create a database using a name that exists as a database alias, even if there is no database with that name. • Database, database alias and instance names can have up to 8 bytes. • On Windows, no instance can have the same name as a service name. **Note:** To avoid potential problems, do not use the special characters @, #, and $ in a database name if you intend to use the database in a communications environment. Also, because these characters are not common to all keyboards, do not use them if you plan to use the database in another language.

Table 3. Database object naming rules

Objects	Guidelines
• Aliases • Buffer pools • Columns • Event monitors • Indexes • Methods • Nodegroups • Packages • Package versions • Schemas • Stored procedures • Tables • Table spaces • Triggers • UDFs • UDTs • Views	Can contain up to 18 bytes *except* for the following: • Table names (including view names, summary table names, alias names, and correlation names), which can contain up to 128 bytes • Column names can contain up to 30 bytes • Package names, which can contain up to 8 bytes • Schema names, which can contain up to 30 bytes • Package versions, which can contain up to 64 bytes • Object names can also include: – valid accented characters (such as ö) – multibyte characters, except multibyte spaces (for multibyte environments) • Package names and package versions can also include periods (.), hyphens (-), and colons (:).

Table 4. Federated database object naming rules

Objects	Guidelines
• Function mappings • Index specifications • Nicknames • Servers • Type mappings • User mappings • Wrappers	• Nicknames, mappings, index specifications, servers, and wrapper names cannot exceed 128 bytes. • Server and nickname options and option settings are limited to 255 bytes. • Names for federated database objects can also include: – Valid accented letters (such as ö) – Multibyte characters, except multibyte spaces (for multibyte environments)

Delimited identifiers and object names:

Keywords can be used. If a keyword is used in a context where it could also be interpreted as an SQL keyword, it must be specified as a delimited identifier.

Using delimited identifiers, it is possible to create an object that violates these naming rules; however, subsequent use of the object could result in errors. For example, if you create a column with a + or – sign included in the name and you subsequently use that column in an index, you will experience problems when you attempt to reorganize the table.

Additional schema names information:

- User-defined types (UDTs) cannot have schema names longer than 8 bytes.
- The following schema names are reserved words and must not be used: SYSCAT, SYSFUN, SYSIBM, SYSSTAT.
- To avoid potential migration problems in the future, do not use schema names that begin with SYS. The database manager will not allow you to create triggers, user-defined types or user-defined functions using a schema name beginning with SYS.
- It is recommended that you not use SESSION as a schema name. Declared temporary tables must be qualified by SESSION. It is therefore possible to have an application declare a temporary table with a name identical to that of a persistent table, in which case the application logic can become overly complicated. Avoid the use of the schema SESSION, except when dealing with declared temporary tables.

Related concepts:

- "General naming rules" on page 23

User, user ID and group naming rules

Table 5. User, user ID and group naming rules

Objects	Guidelines
• Group names • User names • User IDs	• Group names can contain up to 30 characters. • User IDs on Linux and UNIX operating systems can contain up to 8 characters. • User names on Windows can contain up to 30 characters. • When not using Client authentication, non-Windows 32-bit clients connecting to Windows with user names longer than 8 characters are supported when the user name and password are specified explicitly. • Names and IDs cannot: – Be USERS, ADMINS, GUESTS, PUBLIC, LOCAL or any SQL reserved word – Begin with IBM, SQL or SYS.

Notes:

1. Some operating systems allow case sensitive user IDs and passwords. You should check your operating system documentation to see if this is the case.
2. The authorization ID returned from a successful CONNECT or ATTACH is truncated to 8 characters. An ellipsis (...) is appended to the authorization ID and the SQLWARN fields contain warnings to indicate truncation.
3. Trailing blanks from user IDs and passwords are removed.

Related concepts:

- "Federated database object naming rules" in *Administration Guide: Implementation*
- "General naming rules" on page 23

Workstation naming rules

A *workstation name* specifies the NetBIOS name for a database server, database client, or DB2 Personal Edition that resides on the local workstation. This name is stored in the database manager configuration file. The workstation name is known as the *workstation nname*.

In addition, the name you specify:

- Can contain 1 to 8 characters
- Cannot include &, #, or @
- Must be unique within the network

In a partitioned database system, there is still only one workstation *nname* that represents the entire partitioned database system, but each node has its own derived unique NetBIOS *nname*.

The workstation *nname* that represents the partitioned database system is stored in the database manager configuration file for the database partition server that owns the instance.

Each node's unique *nname* is a derived combination of the workstation *nname* and the node number.

If a node does not own an instance, its NetBIOS *nname* is derived as follows:

1. The first character of the instance-owning machine's workstation *nname* is used as the first character of the node's NetBIOS *nname*.
2. The next 1 to 3 characters represent the node number. The range is from 1 to 999.
3. The remaining characters are taken from the instance-owning machine's workstation *nname*. The number of remaining characters depends on the length of the instance-owning machine's workstation *nname*. This number can be from 0 to 4.

For example:

Instance-Owning Machine's *Workstation nname*	**Node Number**	**Derived Node NetBIOS** *nname*
GEORGE	3	G3ORGE
A	7	A7
B2	94	B942
N0076543	21	N216543
GEORGE5	1	G1RGE5

If you have changed the default workstation *nname* during the installation, the workstation *nname*'s last 4 characters should be unique across the NetBIOS network to minimize the chance of deriving a conflicting NetBIOS *nname*.

Related concepts:

- "General naming rules" on page 23

Naming rules in an NLS environment

The basic character set that can be used in database names consists of the single-byte uppercase and lowercase Latin letters (A...Z, a...z), the Arabic numerals (0...9) and the underscore character (_). This list is augmented with three special characters (#, @, and $) to provide compatibility with host database products. Use special characters #, @, and $ with care in an NLS environment because they are not included in the NLS host (EBCDIC) invariant character set. Characters from the extended character set can also be used, depending on the code page that is being used. If you are using the database in a multiple code page environment, you must ensure that all code pages support any elements from the extended character set you plan to use.

When naming database objects (such as tables and views), program labels, host variables, cursors, and elements from the extended character set (for example, letters with diacritical marks) can also be used. Precisely which characters are available depends on the code page in use.

Extended Character Set Definition for DBCS Identifiers:

In DBCS environments, the extended character set consists of all the characters in the basic character set, plus the following:

- All double-byte characters in each DBCS code page, except the double-byte space, are valid letters.
- The double-byte space is a special character.
- The single-byte characters available in each mixed code page are assigned to various categories as follows:

Category	Valid Code Points within each Mixed Code Page
Digits	x30-39
Letters	x23-24, x40-5A, x61-7A, xA6-DF (A6-DF for code pages 932 and 942 only)
Special Characters	All other valid single-byte character code points

Related concepts:

- "DB2 database object naming rules" on page 24
- "General naming rules" on page 23
- "Workstation naming rules" on page 26

Naming rules in a Unicode environment

In a UCS-2 database, all identifiers are in multibyte UTF-8. Therefore, it is possible to use any UCS-2 character in identifiers where the use of a character in the extended character set (for example, an accented character, or a multibyte character) is allowed by the DB2 database system.

Clients can enter any character that is supported by their environment, and all the characters in the identifiers will be converted to UTF-8 by the database manager. Two points must be taken into account when specifying national language characters in identifiers for a UCS-2 database:

- Each non-ASCII character requires two to four bytes. Therefore, an *n*-byte identifier can only hold somewhere between *n*/4 and *n* characters, depending on the ratio of ASCII to non-ASCII characters. If you have only one or two non-ASCII (for example, accented) characters, the limit is closer to *n* characters, while for an identifier that is completely non-ASCII (for example, in Japanese), only *n*/4 to *n*/3 characters can be used.
- If identifiers are to be entered from different client environments, they should be defined using the common subset of characters available to those clients. For example, if a UCS-2 database is to be accessed from Latin-1, Arabic, and Japanese environments, all identifiers should realistically be limited to ASCII.

Related concepts:

- "DB2 database object naming rules" on page 24
- "General naming rules" on page 23
- "Workstation naming rules" on page 26

Reserved schema names and reserved words

There are restrictions on the use of certain names that are required by the database manager. In some cases, names are reserved, and cannot be used by application programs. In other cases, certain names are not recommended for use by application programs, although their use is not prevented by the database manager.

The reserved schema names are:
- SYSCAT
- SYSFUN
- SYSIBM
- SYSSTAT
- SYSPROC

It is strongly recommended that schema names never begin with the 'SYS' prefix, because 'SYS', by convention, is used to indicate an area that is reserved by the system. No user-defined functions, user-defined types, triggers, or aliases can be placed into a schema whose name starts with 'SYS' (SQLSTATE 42939).

The DB2QP schema and the SYSTOOLS schema are set aside for use by DB2 tools. It is recommended that users not explicitly define objects in these schemas, although their use is not prevented by the database manager.

It is also recommended that SESSION not be used as a schema name. Because declared temporary tables must be qualified by SESSION, it is possible to have an application declare a temporary table with a name that is identical to that of a persistent table, complicating the application logic. To avoid this possibility, do not use the schema SESSION except when dealing with declared temporary tables.

There are no specifically reserved words in DB2 Version 9. Keywords can be used as ordinary identifiers, except in a context where they could also be interpreted as SQL keywords. In such cases, the word must be specified as a delimited identifier. For example, COUNT cannot be used as a column name in a SELECT statement, unless it is delimited.

ISO/ANSI SQL99 and other DB2 database products include reserved words that are not enforced by DB2 Database for Linux, UNIX, and Windows; however, it is recommended that these words not be used as ordinary identifiers, because it reduces portability.

For portability across the DB2 database products, the following should be considered reserved words:

```
ACTIVATE           DISALLOW      LOCALE          RESULT
ADD                DISCONNECT    LOCALTIME       RESULT_SET_LOCATOR
AFTER              DISTINCT      LOCALTIMESTAMP  RETURN
ALIAS              DO            LOCATOR         RETURNS
ALL                DOUBLE        LOCATORS        REVOKE
ALLOCATE           DROP          LOCK            RIGHT
ALLOW              DSSIZE        LOCKMAX         ROLLBACK
ALTER              DYNAMIC       LOCKSIZE        ROUTINE
AND                EACH          LONG            ROW
ANY                EDITPROC      LOOP            ROW_NUMBER
AS                 ELSE          MAINTAINED      ROWNUMBER
ASENSITIVE         ELSEIF        MATERIALIZED    ROWS
ASSOCIATE          ENABLE        MAXVALUE        ROWSET
```

Reserved schema names and reserved words

ASUTIME	ENCODING	MICROSECOND	RRN
AT	ENCRYPTION	MICROSECONDS	RUN
ATTRIBUTES	END	MINUTE	SAVEPOINT
AUDIT	END-EXEC	MINUTES	SCHEMA
AUTHORIZATION	ENDING	MINVALUE	SCRATCHPAD
AUX	ERASE	MODE	SCROLL
AUXILIARY	ESCAPE	MODIFIES	SEARCH
BEFORE	EVERY	MONTH	SECOND
BEGIN	EXCEPT	MONTHS	SECONDS
BETWEEN	EXCEPTION	NEW	SECQTY
BINARY	EXCLUDING	NEW_TABLE	SECURITY
BUFFERPOOL	EXCLUSIVE	NEXTVAL	SELECT
BY	EXECUTE	NO	SENSITIVE
CACHE	EXISTS	NOCACHE	SEQUENCE
CALL	EXIT	NOCYCLE	SESSION
CALLED	EXPLAIN	NODENAME	SESSION_USER
CAPTURE	EXTERNAL	NODENUMBER	SET
CARDINALITY	EXTRACT	NOMAXVALUE	SIGNAL
CASCADED	FENCED	NOMINVALUE	SIMPLE
CASE	FETCH	NONE	SOME
CAST	FIELDPROC	NOORDER	SOURCE
CCSID	FILE	NORMALIZED	SPECIFIC
CHAR	FINAL	NOT	SQL
CHARACTER	FOR	NULL	SQLID
CHECK	FOREIGN	NULLS	STACKED
CLOSE	FREE	NUMPARTS	STANDARD
CLUSTER	FROM	OBID	START
COLLECTION	FULL	OF	STARTING
COLLID	FUNCTION	OLD	STATEMENT
COLUMN	GENERAL	OLD_TABLE	STATIC
COMMENT	GENERATED	ON	STAY
COMMIT	GET	OPEN	STOGROUP
CONCAT	GLOBAL	OPTIMIZATION	STORES
CONDITION	GO	OPTIMIZE	STYLE
CONNECT	GOTO	OPTION	SUBSTRING
CONNECTION	GRANT	OR	SUMMARY
CONSTRAINT	GRAPHIC	ORDER	SYNONYM
CONTAINS	GROUP	OUT	SYSFUN
CONTINUE	HANDLER	OUTER	SYSIBM
COUNT	HASH	OVER	SYSPROC
COUNT_BIG	HASHED_VALUE	OVERRIDING	SYSTEM
CREATE	HAVING	PACKAGE	SYSTEM_USER
CROSS	HINT	PADDED	TABLE
CURRENT	HOLD	PAGESIZE	TABLESPACE
CURRENT_DATE	HOUR	PARAMETER	THEN
CURRENT_LC_CTYPE	HOURS	PART	TIME
CURRENT_PATH	IDENTITY	PARTITION	TIMESTAMP
CURRENT_SCHEMA	IF	PARTITIONED	TO
CURRENT_SERVER	IMMEDIATE	PARTITIONING	TRANSACTION
CURRENT_TIME	IN	PARTITIONS	TRIGGER
CURRENT_TIMESTAMP	INCLUDING	PASSWORD	TRIM
CURRENT_TIMEZONE	INCLUSIVE	PATH	TYPE
CURRENT_USER	INCREMENT	PIECESIZE	UNDO
CURSOR	INDEX	PLAN	UNION
CYCLE	INDICATOR	POSITION	UNIQUE
DATA	INHERIT	PRECISION	UNTIL
DATABASE	INNER	PREPARE	UPDATE
DATAPARTITIONNAME	INOUT	PREVVAL	USAGE
DATAPARTITIONNUM	INSENSITIVE	PRIMARY	USER
DATE	INSERT	PRIQTY	USING
DAY	INTEGRITY	PRIVILEGES	VALIDPROC
DAYS	INTERSECT	PROCEDURE	VALUE
DB2GENERAL	INTO	PROGRAM	VALUES
DB2GENRL	IS	PSID	VARIABLE
DB2SQL	ISOBID	QUERY	VARIANT
DBINFO	ISOLATION	QUERYNO	VCAT
DBPARTITIONNAME	ITERATE	RANGE	VERSION

DBPARTITIONNUM	JAR	RANK	VIEW
DEALLOCATE	JAVA	READ	VOLATILE
DECLARE	JOIN	READS	VOLUMES
DEFAULT	KEY	RECOVERY	WHEN
DEFAULTS	LABEL	REFERENCES	WHENEVER
DEFINITION	LANGUAGE	REFERENCING	WHERE
DELETE	LATERAL	REFRESH	WHILE
DENSE_RANK	LC_CTYPE	RELEASE	WITH
DENSERANK	LEAVE	RENAME	WITHOUT
DESCRIBE	LEFT	REPEAT	WLM
DESCRIPTOR	LIKE	RESET	WRITE
DETERMINISTIC	LINKTYPE	RESIGNAL	XMLELEMENT
DIAGNOSTICS	LOCAL	RESTART	YEAR
DISABLE	LOCALDATE	RESTRICT	YEARS

The following list contains the ISO/ANSI SQL2003 reserved words that are not in the previous list:

ABS	GROUPING	REGR_INTERCEPT
ARE	INT	REGR_R2
ARRAY	INTEGER	REGR_SLOPE
ASYMMETRIC	INTERSECTION	REGR_SXX
ATOMIC	INTERVAL	REGR_SXY
AVG	LARGE	REGR_SYY
BIGINT	LEADING	ROLLUP
BLOB	LN	SCOPE
BOOLEAN	LOWER	SIMILAR
BOTH	MATCH	SMALLINT
CEIL	MAX	SPECIFICTYPE
CEILING	MEMBER	SQLEXCEPTION
CHAR_LENGTH	MERGE	SQLSTATE
CHARACTER_LENGTH	METHOD	SQLWARNING
CLOB	MIN	SQRT
COALESCE	MOD	STDDEV_POP
COLLATE	MODULE	STDDEV_SAMP
COLLECT	MULTISET	SUBMULTISET
CONVERT	NATIONAL	SUM
CORR	NATURAL	SYMMETRIC
CORRESPONDING	NCHAR	TABLESAMPLE
COVAR_POP	NCLOB	TIMEZONE_HOUR
COVAR_SAMP	NORMALIZE	TIMEZONE_MINUTE
CUBE	NULLIF	TRAILING
CUME_DIST	NUMERIC	TRANSLATE
CURRENT_DEFAULT_TRANSFORM_GROUP	OCTET_LENGTH	TRANSLATION
CURRENT_ROLE	ONLY	TREAT
CURRENT_TRANSFORM_GROUP_FOR_TYPE	OVERLAPS	TRUE
DEC	OVERLAY	UESCAPE
DECIMAL	PERCENT_RANK	UNKNOWN
DEREF	PERCENTILE_CONT	UNNEST
ELEMENT	PERCENTILE_DISC	UPPER
EXEC	POWER	VAR_POP
EXP	REAL	VAR_SAMP
FALSE	RECURSIVE	VARCHAR
FILTER	REF	VARYING
FLOAT	REGR_AVGX	WIDTH_BUCKET
FLOOR	REGR_AVGY	WINDOW
FUSION	REGR_COUNT	WITHIN

Reserved schema names and reserved words

Chapter 7. Naming conventions

The following conventions apply when naming database manager objects, such as databases and tables:

- Character strings that represent names of database manager objects can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
- Unless otherwise noted, names can be entered in lowercase letters; however, the database manager processes them as if they were uppercase.

  The exception to this convention is character strings that represent names under the systems network architecture (SNA). Many values are case sensitive, such as logical unit names (partner_lu and local_lu). The name must be entered exactly as it appears in the SNA definitions that correspond to those terms.
- A database name or database alias is a unique character string containing from one to eight letters, numbers, or keyboard characters from the set described above.

  Databases are cataloged in the system and local database directories by their aliases in one field, and their original name in another. For most functions, the database manager uses the name entered in the alias field of the database directories. (The exceptions are CHANGE DATABASE COMMENT and CREATE DATABASE, where a directory path must be specified.)
- The name or the alias name of a table or a view is an SQL identifier that is a unique character string 1 to 128 characters in length. Column names can be 1 to 30 characters in length.

  A fully qualified table name consists of the *schema.tablename*. The schema is the unique user ID under which the table was created. The schema name for a declared temporary table must be SESSION.
- Local aliases for remote nodes that are to be cataloged in the node directory cannot exceed eight characters in length.
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.

The following conventions apply when naming user IDs and authentication IDs:

- Character strings that represent names of database manager objects can contain any of the following: a-z, A-Z, 0-9, @, #, and $.
- User IDs and groups may also contain any of the following additional characters when supported by the security plug-in: _, !, %, (,), {, }, –, ., ^.
- User IDs and groups containing any of the following characters must be delimited with quotations when entered through the command line processor: !, %, (,), {, }, –, ., ^,
- The first character in the string must be an alphabetic character, @, #, or $; it cannot be a number or the letter sequences SYS, DBM, or IBM.
- Authentication IDs cannot exceed 30 characters on Windows 32-bit operating systems and 8 characters on all other operating systems.
- Group IDs cannot exceed 30 characters in length.

Related reference:

- "CREATE DATABASE " on page 461
- "CREATE TOOLS CATALOG command" in *Command Reference*

Part 2. Partitioned database environments

Chapter 8. Parallel database systems

Partitioned database environments

DB2 Database for Linux, UNIX, and Windows extends the database manager to the parallel, multi-partition environment. A *database partition* is a part of a database that consists of its own data, indexes, configuration files, and transaction logs. A database partition is sometimes called a node or a database node. A partitioned database environment is a database installation that supports the distribution of data across database partitions.

A *single-partition database* is a database having only one database partition. All data in the database is stored in that single database partition. In this case database partition groups, while present, provide no additional capability.

A *multi-partition database* is a database with two or more database partitions. Tables can be located in one or more database partitions. When a table is in a database partition group consisting of multiple database partitions, some of its rows are stored in one database partition, and other rows are stored in other database partitions.

Usually, a single database partition exists on each physical machine, and the processors on each system are used by the database manager at each database partition to manage its part of the total data in the database.

Because data is distributed across database partitions, you can use the power of multiple processors on multiple physical machines to satisfy requests for information. Data retrieval and update requests are decomposed automatically into sub-requests, and executed in parallel among the applicable database partitions. The fact that databases are split across database partitions is transparent to users issuing SQL statements.

User interaction occurs through one database partition, known as the *coordinator partition* for that user. The coordinator partition runs on the same database partition as the application, or in the case of a remote application, the database partition to which that application is connected. Any database partition can be used as a coordinator partition.

DB2 allows you to store data across several database partitions in the database. This means that the data is physically stored across more than one database partition, and yet can be accessed as though it were located in the same place. Applications and users accessing data in a multi-partition database do not need to be aware of the physical location of the data.

The data, while physically split, is used and managed as a logical whole. Users can choose how to distribute their data by declaring distribution keys. Users can also determine across which and over how many database partitions their data is distributed by selecting the table space and the associated database partition group in which the data should be stored. Suggestions for distribution and replication can be done using the DB2 Design Advisor. In addition, an updatable distribution map is used with a hashing algorithm to specify the mapping of distribution key values to database partitions, which determines the placement and retrieval of each row of data. As a result, you can spread the workload across a multi-partition database for large tables, while allowing smaller tables to be stored on one or more database partitions. Each database partition has local indexes on the data it stores, resulting in increased performance for local data access.

You are not restricted to having all tables divided across all database partitions in the database. DB2 supports *partial declustering*, which means that you can divide tables and their table spaces across a subset of database partitions in the system.

An alternative to consider when you want tables to be positioned on each database partition, is to use materialized query tables and then replicate those tables. You can create a materialized query table containing the information that you need, and then replicate it to each database partition.

Parallelism

Components of a task, such as a database query, can be run in parallel to dramatically enhance performance. The nature of the task, the database configuration, and the hardware environment, all determine how the DB2 database product will perform a task in parallel. These considerations are interrelated, and should be considered together when you work on the physical and logical design of a database. The following types of parallelism are supported by the DB2 database system:
- I/O
- Query
- Utility

Input/output parallelism

When there are multiple containers for a table space, the database manager can exploit *parallel I/O*. Parallel I/O refers to the process of writing to, or reading from, two or more I/O devices simultaneously; it can result in significant improvements in throughput.

Query parallelism

There are two types of query parallelism: interquery parallelism and intraquery parallelism.

Interquery parallelism refers to the ability of the database to accept queries from multiple applications at the same time. Each query runs independently of the others, but DB2 runs all of them at the same time. DB2 database products have always supported this type of parallelism.

Intraquery parallelism refers to the simultaneous processing of parts of a single query, using either *intrapartition parallelism*, *interpartition parallelism*, or both.

Intrapartition parallelism

Intrapartition parallelism refers to the ability to break up a query into multiple parts. Some DB2 utilities also perform this type of parallelism.

Intrapartition parallelism subdivides what is usually considered a single database operation such as index creation, database loading, or SQL queries into multiple parts, many or all of which can be run in parallel *within a single database partition*.

Figure 6 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion. The pieces are copies of each other. To utilize intrapartition parallelism, you must configure the database appropriately. You can choose the degree of parallelism or let the system do it for you. The degree of parallelism represents the number of pieces of a query running in parallel.

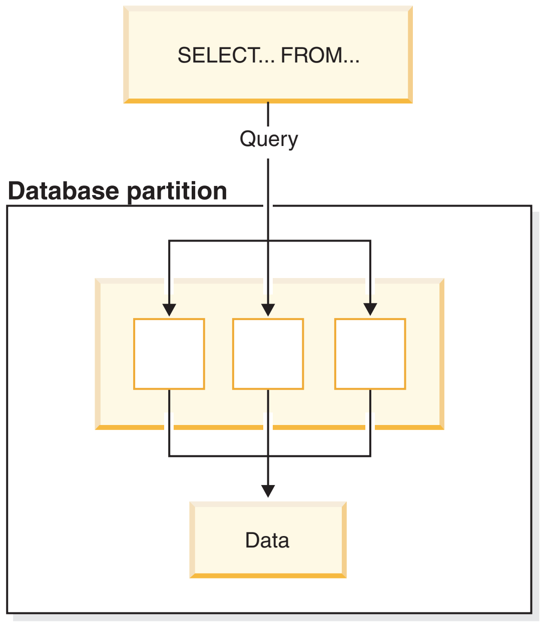


Figure 6. Intrapartition parallelism

Interpartition parallelism

Interpartition parallelism refers to the ability to break up a query into multiple parts across multiple partitions of a partitioned database, on one machine or multiple machines. The query is run in parallel. Some DB2 utilities also perform this type of parallelism.

Interpartition parallelism subdivides what is usually considered a single database operation such as index creation, database loading, or SQL queries into multiple parts, many or all of which can be run in parallel *across multiple partitions of a partitioned database on one machine or on multiple machines*.

Figure 7 on page 40 shows a query that is broken into four pieces that can be run in parallel, with the results returned more quickly than if the query were run in serial fashion on a single database partition.

The degree of parallelism is largely determined by the number of database partitions you create and how you define your database partition groups.

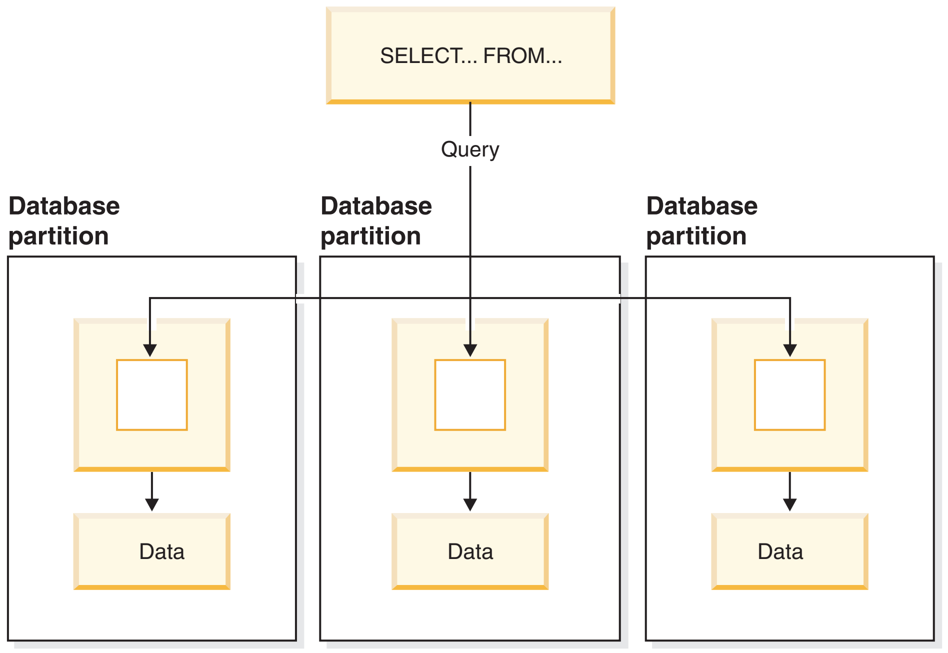


Figure 7. Interpartition parallelism

Simultaneous intrapartition and interpartition parallelism

You can use intrapartition parallelism and interpartition parallelism at the same time. This combination provides two dimensions of parallelism, resulting in an even more dramatic increase in the speed at which queries are processed.

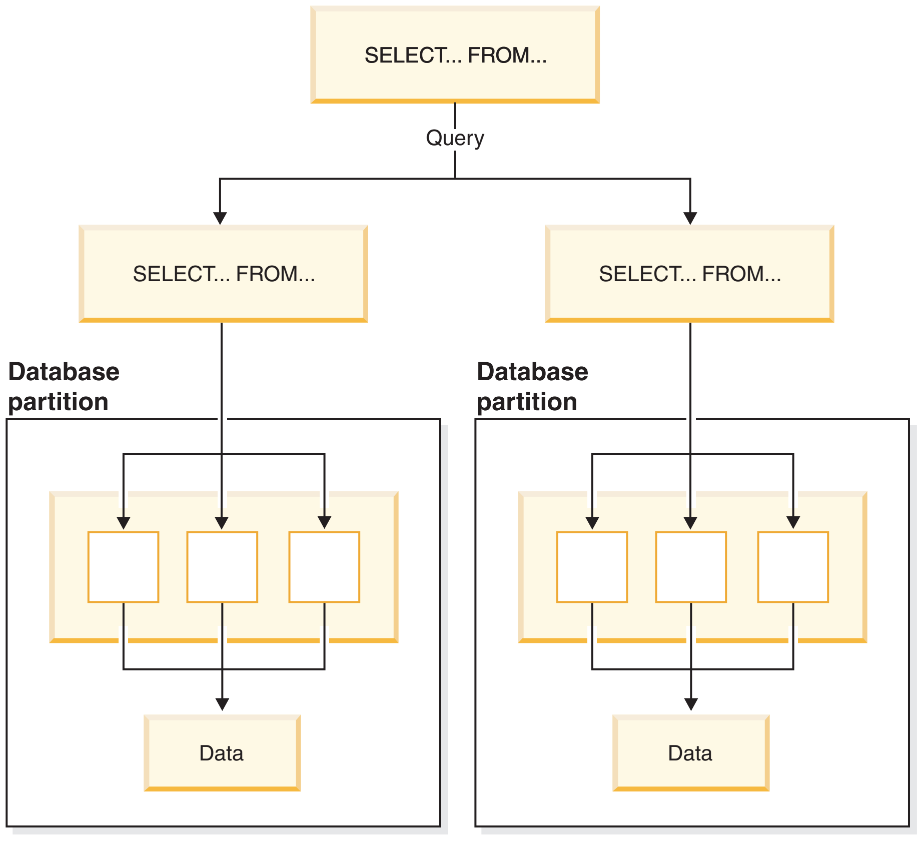


Figure 8. Simultaneous interpartition and intrapartition parallelism

Utility parallelism

DB2 utilities can take advantage of intrapartition parallelism. They can also take advantage of interpartition parallelism; where multiple database partitions exist, the utilities execute in each of the database partitions in parallel.

The load utility can take advantage of intrapartition parallelism and I/O parallelism. Loading data is a CPU-intensive task. The load utility takes advantage of multiple processors for tasks such as parsing and formatting data. It can also use parallel I/O servers to write the data to containers in parallel.

In a partitioned database environment, the LOAD command takes advantage of intrapartition, interpartition, and I/O parallelism by parallel invocations at each database partition where the table resides.

During index creation, the scanning and subsequent sorting of the data occurs in parallel. The DB2 system exploits both I/O parallelism and intrapartition parallelism when creating an index. This helps to speed up index creation when a CREATE INDEX statement is issued, during restart (if an index is marked invalid), and during the reorganization of data.

Backing up and restoring data are heavily I/O-bound tasks. The DB2 system exploits both I/O parallelism and intrapartition parallelism when performing backup and restore operations. Backup exploits I/O parallelism by reading from multiple table space containers in parallel, and asynchronously writing to multiple backup media in parallel.

Related concepts:
- "Database partition and processor environments" on page 42

Database partition and processor environments

This section provides an overview of the following hardware environments:
- Single database partition on a single processor (uniprocessor)
- Single database partition with multiple processors (SMP)
- Multiple database partition configurations
  - Database partitions with one processor (MPP)
  - Database partitions with multiple processors (cluster of SMPs)
  - Logical database partitions

Capacity and scalability are discussed for each environment. *Capacity* refers to the number of users and applications able to access the database. This is in large part determined by memory, agents, locks, I/O, and storage management. *Scalability* refers to the ability of a database to grow and continue to exhibit the same operating characteristics and response times.

Single database partition on a single processor

This environment is made up of memory and disk, but contains only a single CPU (see Figure 9). It is referred to by many different names, including stand-alone database, client/server database, serial database, uniprocessor system, and single node or non-parallel environment.

The database in this environment serves the needs of a department or small office, where the data and system resources (including a single processor or CPU) are managed by a single database manager.

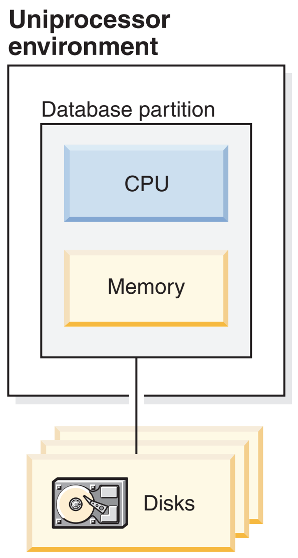


Figure 9. Single database partition on a single processor

Capacity and scalability

In this environment you can add more disks. Having one or more I/O servers for each disk allows for more than one I/O operation to take place at the same time.

A single-processor system is restricted by the amount of disk space the processor can handle. As workload increases, a single CPU may not be able to process user requests any faster, regardless of other components, such as memory or disk, that you may add. If you have reached maximum capacity or scalability, you can consider moving to a single database partition system with multiple processors.

Single database partition with multiple processors

This environment is typically made up of several equally powerful processors within the same machine (see Figure 10), and is called a *symmetric multiprocessor (SMP)* system. Resources, such as disk space and memory, are *shared*.

With multiple processors available, different database operations can be completed more quickly. DB2 database systems can also divide the work of a single query among available processors to improve processing speed. Other database operations, such as loading data, backing up and restoring table spaces, and creating indexes on existing data, can take advantage of multiple processors.

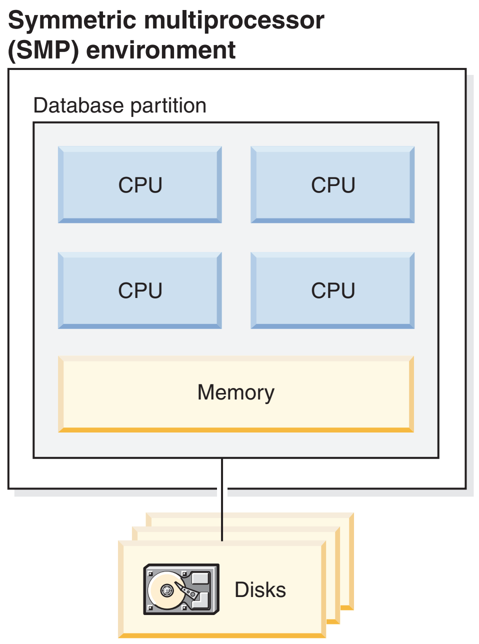


Figure 10. Single partition database symmetric multiprocessor environment

Capacity and scalability

In this environment you can add more processors. However, since the different processors may attempt to access the same data, limitations with this environment can appear as your business operations grow. With shared memory and shared disks, you are effectively sharing all of the database data.

You can increase the I/O capacity of the database partition associated with your processor by increasing the number of disks. You can establish I/O servers to specifically deal with I/O requests. Having one or more I/O servers for each disk allows for more than one I/O operation to take place at the same time.

If you have reached maximum capacity or scalability, you can consider moving to a system with multiple database partitions.

Multiple database partition configurations

You can divide a database into multiple database partitions, each on its own machine. Multiple machines with multiple database partitions can be grouped together. This section describes the following database partition configurations:

- Database partitions on systems with one processor
- Database partitions on systems with multiple processors
- Logical database partitions

Database partitions with one processor

In this environment, there are many database partitions. Each database partition resides on its own machine, and has its own processor, memory, and disks (Figure 11). All the machines are connected by a communications facility. This environment is referred to by many different names, including: cluster, cluster of uniprocessors, massively parallel processing (MPP) environment, and shared-nothing configuration. The latter name accurately reflects the arrangement of resources in this environment. Unlike an SMP environment, an MPP environment has no shared memory or disks. The MPP environment removes the limitations introduced through the sharing of memory and disks.

A partitioned database environment allows a database to remain a logical whole, despite being physically divided across more than one database partition. The fact that data is distributed remains transparent to most users. Work can be divided among the database managers; each database manager in each database partition works against its own part of the database.

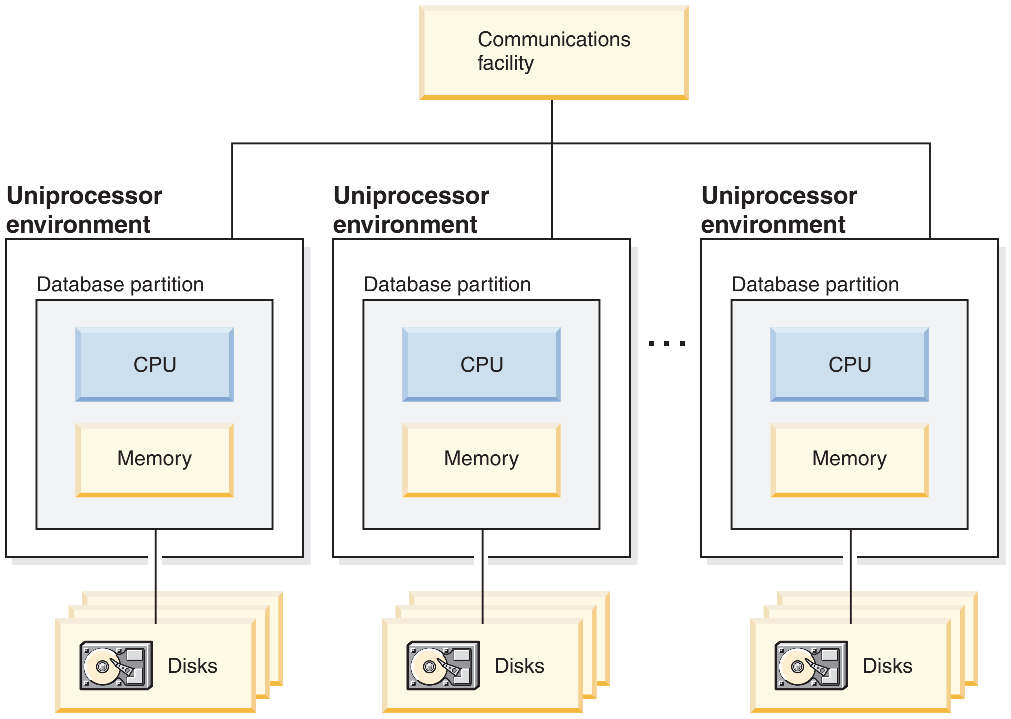


Figure 11. Massively parallel processing (MPP) environment

Capacity and scalability: In this environment you can add more database partitions (nodes) to your configuration. On some platforms, for example the RS/6000® SP™, the maximum number is 512 nodes. However, there may be practical limits on managing a high number of machines and instances.

If you have reached maximum capacity or scalability, you can consider moving to a system where each database partition has multiple processors.

Database partitions with multiple processors

An alternative to a configuration in which each database partition has a single processor, is a configuration in which each database partition has multiple processors. This is known as an *SMP cluster* (Figure 12).

This configuration combines the advantages of SMP and MPP parallelism. This means that a query can be performed in a single database partition across multiple processors. It also means that a query can be performed in parallel across multiple database partitions.

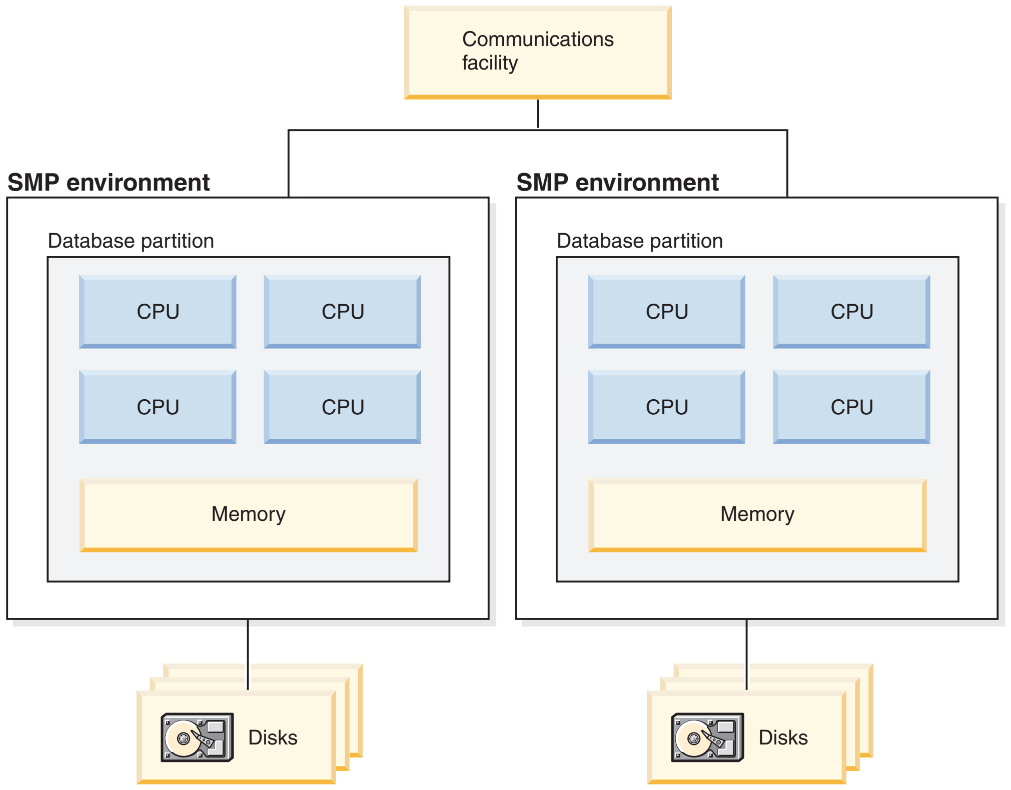


Figure 12. Several symmetric multiprocessor (SMP) environments in a cluster

Capacity and scalability: In this environment you can add more database partitions, and you can add more processors to existing database partitions.

Logical database partitions

A logical database partition differs from a physical partition in that it is not given control of an entire machine. Although the machine has shared resources, database partitions do not share the resources. Processors are shared but disks and memory are not.

Logical database partitions provide scalability. Multiple database managers running on multiple logical partitions may make fuller use of available resources than a single database manager could. Figure 13 illustrates the fact that you may gain more scalability on an SMP machine by adding more database partitions; this is particularly true for machines with many processors. By distributing the database, you can administer and recover each database partition separately.

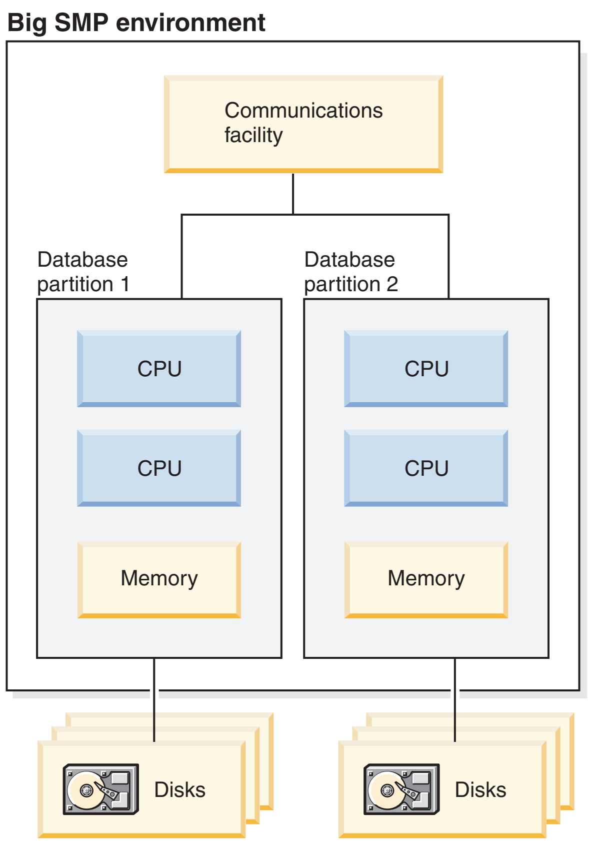


Figure 13. Partitioned database with symmetric multiprocessor environment

Figure 14 on page 47 illustrates that you can multiply the configuration shown in Figure 13 to increase processing power.

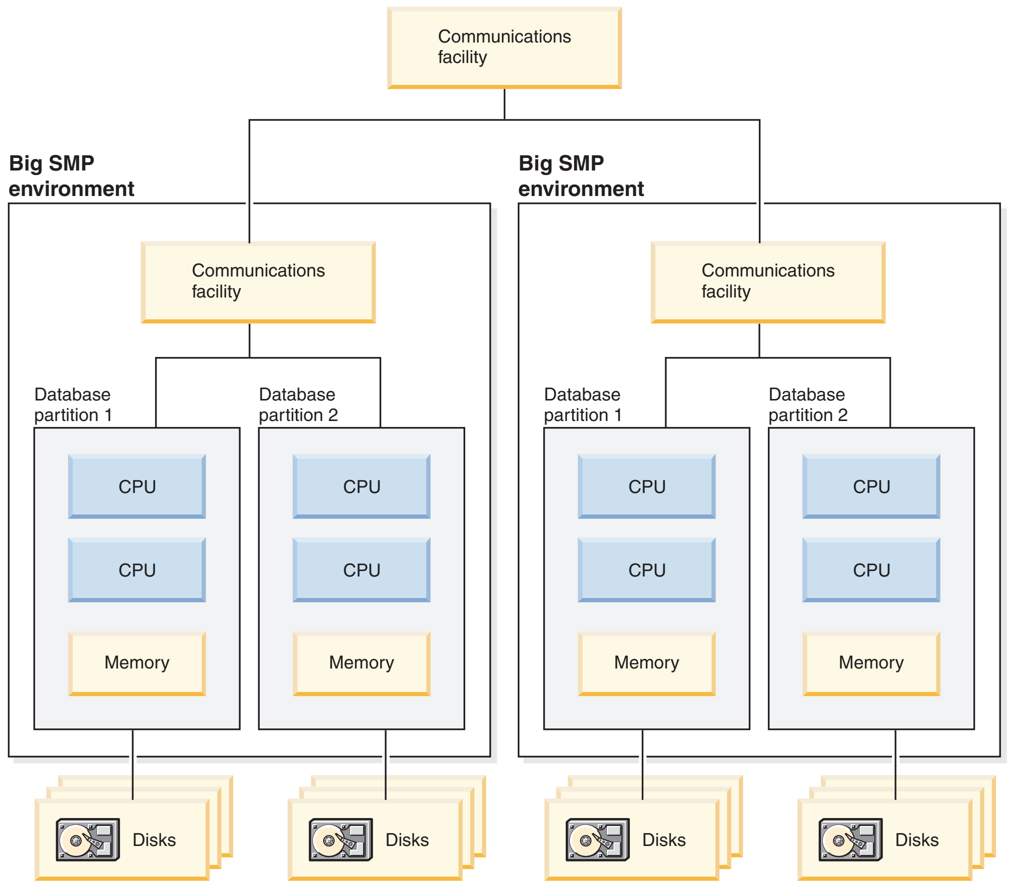


Figure 14. Partitioned database with symmetric multiprocessor environments clustered together

Note: The ability to have two or more database partitions coexist on the same machine (regardless of the number of processors) allows greater flexibility in designing high availability configurations and failover strategies. Upon machine failure, a database partition can be automatically moved and restarted on a second machine that already contains another database partition of the same database.

Summary of parallelism best suited to each hardware environment

The following table summarizes the types of parallelism best suited to take advantage of the various hardware environments.

Table 6. Types of Parallelism Possible in Each Hardware Environment

Hardware Environment	**I/O Parallelism**	**Intra-Query Parallelism**	
		Intra-Partition Parallelism	**Inter-Partition Parallelism**
Single Database Partition, Single Processor	Yes	No(1)	No
Single Database Partition, Multiple Processors (SMP)	Yes	Yes	No
Multiple Database Partitions, One Processor (MPP)	Yes	No(1)	Yes

Table 6. Types of Parallelism Possible in Each Hardware Environment (continued)

Hardware Environment	**I/O Parallelism**	**Intra-Query Parallelism**	
		Intra-Partition Parallelism	**Inter-Partition Parallelism**
Multiple Database Partitions, Multiple Processors (cluster of SMPs)	Yes	Yes	Yes
Logical Database Partitions	Yes	Yes	Yes
Note: (1) There may be an advantage to setting the degree of parallelism (using one of the configuration parameters) to some value greater than one, even on a single processor system, especially if the queries you execute are not fully utilizing the CPU (for example, if they are I/O bound).			

Related concepts:

- "Parallelism" on page 38

Chapter 9. Database partitioning across multiple database partitions

DB2 allows great flexibility in spreading data across multiple database partitions (nodes) of a partitioned database. Users can choose how to distribute their data by declaring distribution keys, and can determine which and how many database partitions their table data can be spread across by selecting the database partition group and table space in which the data should be stored. In addition, a distribution map (which is updatable) specifies the mapping of distribution key values to database partitions. This makes it possible for flexible workload parallelization across a partitioned database for large tables, while allowing smaller tables to be stored on one or a small number of database partitions if the application designer so chooses. Each local database partition may have local indexes on the data it stores to provide high performance local data access.

In a partitioned database, the distribution key is used to distribute table data across a set of database partitions. Index data is also partitioned with its corresponding tables, and stored locally at each database partition.

Before database partitions can be used to store data, they must be defined to the database manager. Database partitions are defined in a file called db2nodes.cfg.

The distribution key for a table in a table space on a partitioned database partition group is specified in the CREATE TABLE statement or the ALTER TABLE statement. If not specified, a distribution key for a table is created by default from the first column of the primary key. If no primary key is defined, the default distribution key is the first column defined in that table that has a data type other than a long or a LOB data type. Tables in partitioned databases must have at least one column that is neither a long nor a LOB data type. A table in a table space that is in a single partition database partition group will have a distribution key only if it is explicitly specified.

Rows are placed in a database partition as follows:

1. A hashing algorithm (database partitioning function) is applied to all of the columns of the distribution key, which results in the generation of a distribution map index value.
2. The database partition number at that index value in the distribution map identifies the database partition in which the row is to be stored.

DB2 supports *partial declustering*, which means that a table can be distributed across a subset of database partitions in the system (that is, a database partition group). Tables do not have to be distributed across all of the database partitions in the system.

DB2 has the capability of recognizing when data being accessed for a join or a subquery is located at the same database partition in the same database partition group. This is known as *table collocation*. Rows in collocated tables with the same distribution key values are located on the same database partition. DB2 can choose to perform join or subquery processing at the database partition in which the data is stored. This can have significant performance advantages.

Collocated tables must:

- Be in the same database partition group, one that is not being redistributed. (During redistribution, tables in the database partition group may be using different distribution maps – they are not collocated.)
- Have distribution keys with the same number of columns.
- Have the corresponding columns of the distribution key be database partition-compatible.
- Be in a single partition database partition group defined on the same database partition.

Related reference:

- "Database partition-compatible data types" in *SQL Reference, Volume 1*

Database partition groups

A database partition group is a set of one or more database partitions defined as belongin to a database. When you want to create tables for the database, you first create the database partition group where the table spaces will be stored, then you create the table space where the tables will be stored.

You can define named subsets of one or more database partitions in a database. Each subset you define is known as a *database partition group*. Each subset that contains more than one database partition is known as a *multipartition database partition group*. Multipartition database partition groups can only be defined with database partitions that belong to the same instance. A database partition group can contain as few as one database partition, or span all of the database partitions in the database.

Figure 19 on page 250 shows an example of a database with five database partitions in which:

- A database partition group spans all but one of the database partitions (Database Partition Group 1).
- A database partition group contains one database partition (Database Partition Group 2).
- A database partition group contains two database partitions. (Database Partition Group 3).
- The database partition within Database Partition Group 2 is shared (and overlaps) with Database Partition Group 1.
- There is a single database partition within Database Partition Group 3 that is shared (and overlaps) with Database Partition Group 1.

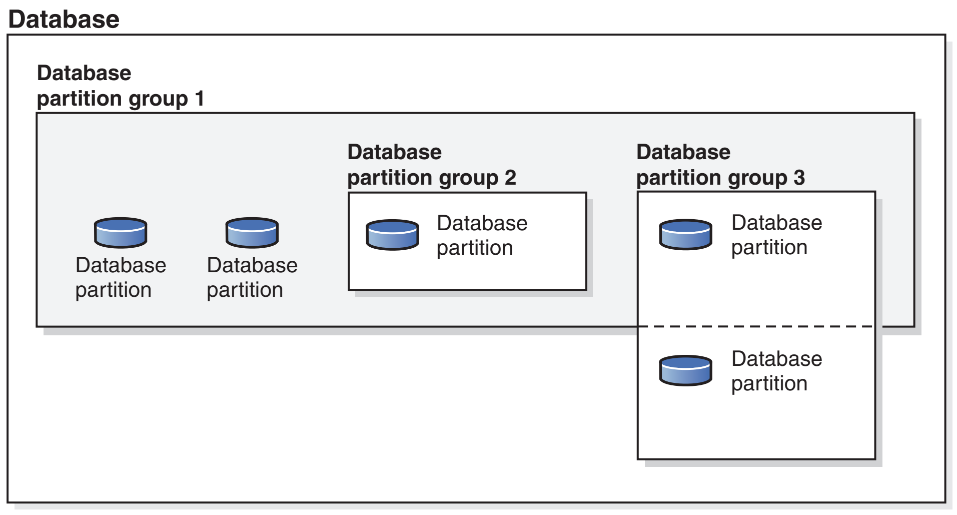


Figure 15. Database partition groups in a database

You create a new database partition group using the CREATE DATABASE PARTITION GROUP statement. You can modify it using the ALTER DATABASE PARTITION GROUP statement. Data is divided across all the database partitions in a database partition group, and you can add or drop one or more database partitions from a database partition group. If you are using a multipartition database partition group, you must look at several database partition group design considerations.

Each database partition that is part of the database system configuration must already be defined in a *database partition configuration file* called `db2nodes.cfg`. A database partition group can contain as few as one database partition, or as many as the entire set of database partitions defined for the database system.

When a database partition group is created or modified, a *distribution map* is associated with it. A distribution map, in conjunction with a *distribution key* and a hashing algorithm, is used by the database manager to determine which database partition in the database partition group will store a given row of data.

In a non-partitioned database, no distribution key or distribution map is required. A database partition is a part of the database, complete with user data, indexes, configuration files, and transaction logs. Default database partition groups that were created when the database was created are used by the database manager. IBMCATGROUP is the default database partition group for the table space containing the system catalogs. IBMTEMPGROUP is the default database partition group for system temporary table spaces. IBMDEFAULTGROUP is the default database partition group for the table spaces containing the user defined tables that you may choose to put there. A user temporary table space for a declared temporary table can be created in IBMDEFAULTGROUP or any user-created database partition group but not in IBMTEMPGROUP.

When working with database partition groups you can:
- Create a database partition group.
- Change the comment associated with a database partition group.
- Add database partitions to a database partition group.
- Drop database partitions from a database partition group.
- Redistribute table data within a database partition group.

Related concepts:
- "Database partition group design" on page 251
- "Distribution keys" on page 254
- "Distribution maps" on page 252

Related reference:
- "ALTER DATABASE PARTITION GROUP " on page 845
- "CREATE DATABASE PARTITION GROUP " on page 918

Chapter 11. Execution parallelism

You must modify configuration parameters to take advantage of parallelism within a database partition or within a non-partitioned database. For example, intra-partition parallelism can be used to take advantage of the multiple processors on a symmetric multi-processor (SMP) machine.

Enabling inter-partition query parallelism

Inter-partition parallelism occurs automatically based on the number of database partitions and the distribution of data across these database partitions.

Note: You must modify configuration parameters to take advantage of parallelism within a database partition or within a non-partitioned database. For example, intra-partition parallelism can be used to take advantage of the multiple processors on a symmetric multi-processor (SMP) machine.

Related concepts:
- "Partitioned database environments" on page 37
- "Database partition group design" on page 251
- "Database partition and processor environments" on page 42
- "Adding database partitions in a partitioned database environment" on page 328

Related tasks:
- "Redistributing data across database partitions" on page 339
- "Enabling database partitioning in a database" on page 286
- "Enabling intra-partition parallelism for queries" in *Administration Guide: Implementation*

Enabling parallelism for loading data

The load utility automatically makes use of parallelism, or you can use the following parameters on the `LOAD` command:
- CPU_PARALLELISM
- DISK_PARALLELISM

In a partitioned database environment, inter-partition parallelism for data loading occurs automatically when the target table is defined on multiple database partitions. Inter-partition parallelism for data loading can be overridden by specifying OUTPUT_DBPARTNUMBS. The load utility also intelligently enables database partitioning parallelism depending on the size of the target database partitions. MAX_NUM_PART_AGENTS can be used to control the maximum degree of parallelism selected by the load utility. Database partitioning parallelism can be overridden by specifying PARTITIONING_DBPARTNUMS when ANYORDER is also specified.

Related concepts:
- "Load overview" in *Data Movement Utilities Guide and Reference*

- Chapter 35, “Load in a partitioned database environment - overview,” on page 431

Enabling parallelism when creating indexes

To enable parallelism when creating an index:

- The *intra_parallel* database manager configuration parameter must be ON
- The table must be large enough to benefit from parallelism
- Multiple processors must be enabled on an SMP computer.

Related reference:

- “CREATE INDEX ” on page 921
- “intra_parallel - Enable intra-partition parallelism configuration parameter” in *Performance Guide*

Enabling I/O parallelism when backing up a database or table space

To enable I/O parallelism when backing up a database or table space:

- Use more than one target media.
- Configure table spaces for parallel I/O by defining multiple containers, or use a single container with multiple disks, and the appropriate use of the DB2_PARALLEL_IO registry variable. If you want to take advantage of parallel I/O, you must consider the implications of what must be done before you define any containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to backup your database or table space.
- Use the PARALLELISM parameter on the `BACKUP` command to specify the degree of parallelism.
- Use the WITH `num-buffers` BUFFERS parameter on the `BACKUP` command to ensure enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

  Also, use a backup buffer size that is:

  – As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.

  – At least as large as the largest (extentsize * number of containers) product of the table spaces being backed up.

Related reference:

- “BACKUP DATABASE ” on page 437

Enabling I/O parallelism when restoring a database or table space

To enable I/O parallelism when restoring a database or table space:

- Use more than one source media.
- Configure table spaces for parallel I/O. You must make the decision to use this option before you define your containers. This cannot be done whenever you see a need; it must be planned for before you reach the point where you need to restore your database or table space.

- Use the PARALLELISM parameter on the `RESTORE` command to specify the degree of parallelism.
- Use the WITH `num-buffers` BUFFERS parameter on the `RESTORE` command to ensure enough buffers are available to accommodate the degree of parallelism. The number of buffers should equal the number of target media you have plus the degree of parallelism selected plus a few extra.

  Also, use a restore buffer size that is:
  - As large as feasible. 4 MB or 8 MB (1024 or 2048 pages) is a good rule of thumb.
  - At least as large as the largest (extentsize * number of containers) product of the table spaces being restored.
  - The same as, or an even multiple of, the backup buffer size.

Related reference:
- "RESTORE DATABASE " on page 591

Configuration parameters that affect the number of agents

The following database manager configuration parameters determine how many database agents are created and how they are managed:
- Maximum Number of Agents (*maxagents*): The number of agents that can be working at any one time. This value applies to the total number of agents that are working on all applications, including coordinator agents, subagents, inactive agents, and idle agents.
- Agent Pool Size (*num_poolagents*): The total number of agents, including active agents and agents in the agent pool, that are kept available in the system. The default value for this parameter is half the number specified for *maxagents*.
- Initial Number of Agents in Pool (*num_initagents*): When the database manager is started, a pool of worker agents is created based on this value. This speeds up performance for initial queries. The worker agents all begin as idle agents.
- Maximum Number of Connections (*max_connections*): specifies the maximum number of connections allowed to the database manager system on each database partition.
- Maximum Number of Coordinating Agents (*max_coordagents*): For partitioned database environments and environments with intra-partition parallelism enabled when the Connection Concentrator is enabled, this value limits the number of coordinating agents.
- Maximum Number of Concurrent Agents (*maxcagents*): This value controls the number of *tokens* permitted by the database manager. For each database transaction (unit of work) that occurs when a client is connected to a database, a coordinating agent must obtain permission to process the transaction from the database manager. This permission is called a *processing token*. The database manager permits only agents that have a processing token to execute a unit of work against a database. If a token is not available, the agent must wait until one is available to process the transaction.

  This parameter can be useful in an environment in which peak usage requirements exceed system resources for memory, CPU, and disk. For example, in such an environment, paging might cause performance degradation for peak load periods. You can use this parameter to control the load and avoid performance degradation, although it can affect either concurrency or wait time, or both.

Related concepts:

- "Agents in a partitioned database" in *Performance Guide*
- Chapter 4, "Database agents," on page 19
- Chapter 5, "Database agent management," on page 21
- "Connection concentrator" in *DB2 Connect User's Guide*

Chapter 12. Synchronizing clocks in a partitioned database environment

You should maintain relatively synchronized system clocks across the database partition servers to ensure smooth database operations and unlimited forward recoverability. Time differences among the database partition servers, plus any potential operational and communications delays for a transaction should be less than the value specified for the *max_time_diff* (maximum time difference among nodes) database manager configuration parameter.

To ensure that the log record time stamps reflect the sequence of transactions in a partitioned database environment, DB2 uses the system clock on each machine as the basis for the time stamps in the log records. If, however, the system clock is set ahead, the log clock is automatically set ahead with it. Although the system clock can be set back, the clock for the logs cannot, and remains at the *same* advanced time until the system clock matches this time. The clocks are then in synchrony. The implication of this is that a short term system clock error on a database node can have a long lasting effect on the time stamps of database logs.

For example, assume that the system clock on database partition server A is mistakenly set to November 7, 2005 when the year is 2003, and assume that the mistake is corrected *after* an update transaction is committed in the database partition at that database partition server. If the database is in continual use, and is regularly updated over time, any point between November 7, 2003 and November 7, 2005 is virtually unreachable through rollforward recovery. When the COMMIT on database partition server A completes, the time stamp in the database log is set to 2005, and the log clock remains at November 7, 2005 until the system clock matches this time. If you attempt to roll forward to a point in time within this time frame, the operation will stop at the first time stamp that is beyond the specified stop point, which is November 7, 2003.

Although DB2 cannot control updates to the system clock, the *max_time_diff* database manager configuration parameter reduces the chances of this type of problem occurring:

- The configurable values for this parameter range from 1 minute to 24 hours.
- When the first connection request is made to a non-catalog partition, the database partition server sends its time to the catalog partition for the database. The catalog partition then checks that the time on the database partition requesting the connection, and its own time are within the range specified by the *max_time_diff* parameter. If this range is exceeded, the connection is refused.
- An update transaction that involves more than two database partition servers in the database must verify that the clocks on the participating database partition servers are in synchrony before the update can be committed. If two or more database partition servers have a time difference that exceeds the limit allowed by *max_time_diff*, the transaction is rolled back to prevent the incorrect time from being propagated to other database partition servers.

Related reference:

- “max_time_diff - Maximum time difference among nodes ” on page 1516

Part 3. DB2 security considerations

Chapter 13. Authentications, authorizations, privileges, and authorities

Security

To protect data and resources associated with a database server, DB2 Database for Linux, UNIX, and Windows uses a combination of external security services and internal access control information. To access a database server, you must pass some security checks before you are given access to database data or resources. The first step in database security is called *authentication*, where you must prove that you are who you say you are. The second step is called *authorization*, where the database manager decides if the validated user is allowed to perform the requested action, or access the requested data.

Related concepts:
- "Authentication" on page 61
- "Authorization" on page 62

Authentication

Authentication of a user is completed using a security facility outside of DB2 Database for Linux, UNIX, and Windows. The security facility can be part of the operating system, a separate product or, in certain cases, may not exist at all. On UNIX based systems, the security facility is in the operating system itself.

The security facility requires two items to authenticate a user: a user ID and a password. The user ID identifies the user to the security facility. By supplying the correct password, information known only to the user and the security facility, the user's identity (corresponding to the user ID) is verified.

Once authenticated:

- The user must be identified to DB2 using an SQL authorization name or *authid*. This name can be the same as the user ID, or a mapped value. For example, on UNIX operating systems, a DB2 *authid* is derived by transforming to uppercase letters a UNIX user ID that follows DB2 naming conventions.
- A list of groups to which the user belongs is obtained. Group membership may be used when authorizing the user. Groups are security facility entities that must also map to DB2 authorization names. This mapping is done in a method similar to that used for user IDs.

DB2 V9.1 uses the security facility to authenticate users in one of two ways:

- DB2 uses a successful security system login as evidence of identity, and allows:
  - Use of local commands to access local data
  - Use of remote connections when the server trusts the client authentication.
- DB2 accepts a user ID and password combination. It uses successful validation of this pair by the security facility as evidence of identity and allows:
  - Use of remote connections where the server requires proof of authentication
  - Use of operations where the user wants to run a command under an identity other than the identity used for login.

DB2 on AIX® can log failed password attempts with the operating system, and detect when a client has exceeded the number of allowable login tries, as specified by the LOGINRETRIES parameter.

Related concepts:

- "Authentication methods for your server" on page 89
- "Authorization" on page 62
- "Authorization, privileges, and object ownership" on page 63

Partitioned database authentication considerations

In a partitioned database, each partition of the database must have the same set of users and groups defined. If the definitions are not the same, the user may be authorized to do different things on different partitions. Consistency across all partitions is recommended.

Related concepts:

- "Authentication methods for your server" on page 89

Authorization

Authorization is the process whereby DB2 obtains information about an authenticated DB2 user, indicating the database operations that user may perform, and what data objects may be accessed. With each user request, there may be more than one authorization check, depending on the objects and operations involved.

Authorization is performed using DB2 facilities. DB2 tables and configuration files are used to record the permissions associated with each authorization name. When an authenticated user tries to access data, the authorization name of the user, and those of groups to which the user belongs, are compared with the recorded permissions. Based on this comparison, DB2 decides whether to allow the requested access.

There are three types of permissions recorded by DB2 Database for Linux, UNIX, and Windows: privileges, authority levels, and LBAC credentials.

A *privilege* defines a single permission for an authorization name, enabling a user to create or access database resources. Privileges are stored in the database catalogs.

Authority levels provide a method of grouping privileges and control over higher-level database manager maintenance and utility operations. Database-specific authorities are stored in the database catalogs; system authorities are associated with group membership, and the group names that are associated with the authority levels are stored in the database manager configuration file for a given instance.

LBAC credentials are LBAC security labels and LBAC rule exemptions that allow access to data protected by label-based access control (LBAC). LBAC credentials are stored in the database catalogs.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. Unless otherwise specified, group authorization names can be used anywhere that authorization names are used for authorization purposes. In general, group membership is considered for dynamic SQL and non-database object authorizations (such as instance level commands and utilities), but is not considered for static SQL. The exception to this general case occurs when privileges are granted to PUBLIC: these are considered when static SQL is processed. Specific cases where group membership does not apply are noted throughout the DB2 documentation, where applicable.

Related concepts:
- "Authorization and privileges" in *SQL Reference, Volume 1*
- "Authorization, privileges, and object ownership" on page 63
- "Label-based access control (LBAC) overview" on page 109

Authorization, privileges, and object ownership

Users (identified by an authorization ID) can successfully execute SQL or XQuery statements only if they have the authority to perform the specified function. To create a table, a user must be authorized to create tables; to alter a table, a user must be authorized to alter the table; and so forth.

There are two forms of authorization, *administrative authority* and *privileges*, discussed below.

The database manager requires that each user be specifically authorized, either implicitly or explicitly, to use each database function needed to perform a specific task. *Explicit* authorities or privileges are granted to the user (GRANTEETYPE of U in the database catalogs). *Implicit* authorities or privileges are granted to a group to which the user belongs (GRANTEETYPE of G in the database catalogs).

Administrative authority:

The person or persons holding administrative authority are charged with the task of controlling the database manager and are responsible for the safety and integrity of the data. Those with administrative authority levels of SYSADM and DBADM implicitly have all privileges on all objects except objects pertaining to database security and control who will have access to the database manager and the extent of this access.

Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. *Database authorities* enable users to perform activities at the database level. A user or group can have one or more of the following authorities:

- Administrative authority level that operates at the instance level, SYSADM (system administrator)

  The SYSADM authority level provides control over all the resources created and maintained by the database manager. The system administrator possesses all the authorities of DBADM, SYSCTRL, SYSMAINT, and SYSMON, and the authority to grant and revoke DBADM authority and SECADM authority.

  The user who possesses SYSADM authority is responsible both for controlling the database manager, and for ensuring the safety and integrity of the data. SYSADM authority provides implicit DBADM authority within a database but does not provide implicit SECADM authority within a database.
- Administrative authority levels that operate at the database level:
  - DBADM (database administrator)

    The DBADM authority level applies at the database level and provides administrative authority over a single database. This database administrator possesses the privileges required to create objects, issue database commands, and access table data. The database administrator can also grant and revoke CONTROL and individual privileges.
  - SECADM (security administrator)

    The SECADM authority level applies at the database level and is the authority required to create and drop security label components, security policies, and security labels, which are used to protect tables. It is also the authority required to grant and revoke security labels and exemptions as well as to grant and revoke the SETSESSIONUSER privilege. A user with the SECADM authority can transfer the ownership of objects that they do not own. The SECADM authority has no inherent privilege to access data stored in tables and has no other additional inherent privilege. It can only be granted by a user with SYSADM authority. The SECADM authority can be granted to a user but cannot be granted to a group or to PUBLIC.
- System control authority levels that operate at the instance level:
  - SYSCTRL (system control)

    The SYSCTRL authority level provides control over operations that affect system resources. For example, a user with SYSCTRL authority can create, update, start, stop, or drop a database. This user can also start or stop an instance, but cannot access table data. Users with SYSCTRL authority also have SYSMON authority.
  - SYSMAINT (system maintenance)

The SYSMAINT authority level provides the authority required to perform maintenance operations on all databases associated with an instance. A user with SYSMAINT authority can update the database configuration, backup a database or table space, restore an existing database, and monitor a database. Like SYSCTRL, SYSMAINT does not provide access to table data. Users with SYSMAINT authority also have SYSMON authority.

- The SYSMON (system monitor) authority level

  SYSMON provides the authority required to use the database system monitor. It operates at the instance level.
- Database authorities

  To perform activities such as creating a table or a routine, or for loading data into a table, specific database authorities are required. For example, the LOAD database authority is required for use of the load utility to load data into tables (a user must also have INSERT privilege on the table).

Figure 16 illustrates the relationship between authorities and their span of control (database, database manager).

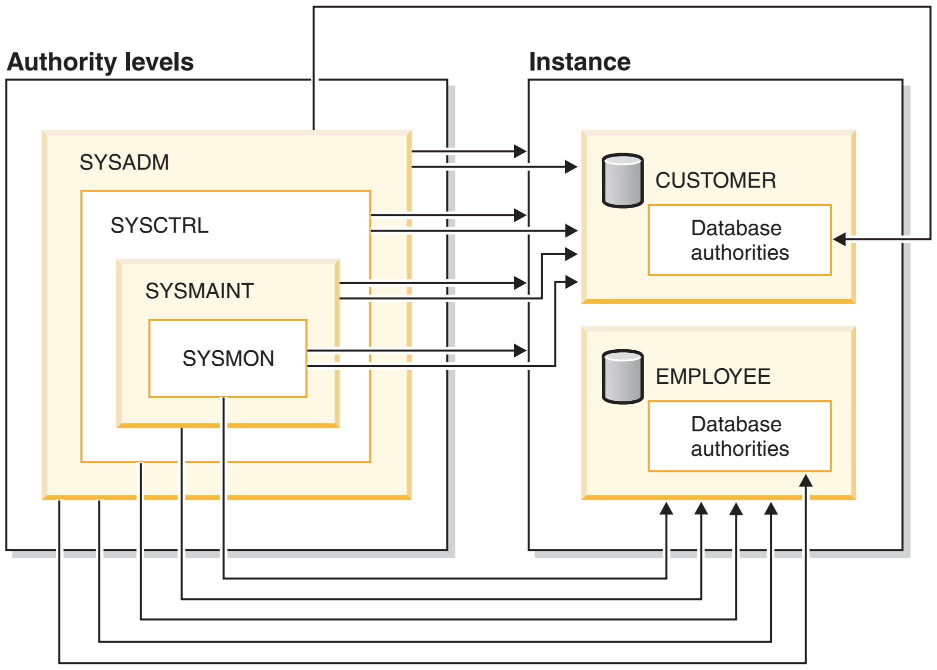


Figure 16. Hierarchy of Authorities

Privileges:

Privileges are those activities that a user is allowed to perform. Authorized users can create objects, have access to objects they own, and can pass on privileges on their own objects to other users by using the GRANT statement.

Privileges may be granted to individual users, to groups, or to PUBLIC. PUBLIC is a special group that consists of all users, including future users. Users that are members of a group will indirectly take advantage of the privileges granted to the group, where groups are supported.

The CONTROL privilege: Possessing the CONTROL privilege on an object allows a user to access that database object, and to grant and revoke privileges to or from other users on that object.

Note: The CONTROL privilege only apples to tables, views, nicknames, indexes, and packages.

If a different user requires the CONTROL privilege to that object, a user with SYSADM or DBADM authority could grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked from the object owner, however, the object owner can be changed by using the TRANSFER OWNERSHIP statement.

In some situations, the creator of an object automatically obtains the CONTROL privilege on that object.

Individual privileges: Individual privileges can be granted to allow a user to carry out specific tasks on specific objects. Users with administrative authority (SYSADM or DBADM) or the CONTROL privilege can grant and revoke privileges to and from users.

Individual privileges and database authorities allow a specific function, but do not include the right to grant the same privileges or authorities to other users. The right to grant table, view, schema, package, routine, and sequence privileges to others can be extended to other users through the WITH GRANT OPTION on the GRANT statement. However, the WITH GRANT OPTION does not allow the person granting the privilege to revoke the privilege once granted. You must have SYSADM authority, DBADM authority, or the CONTROL privilege to revoke the privilege.

Privileges on objects in a package or routine: When a user has the privilege to execute a package or routine, they do not necessarily require specific privileges on the objects used in the package or routine. If the package or routine contains static SQL or XQuery statements, the privileges of the owner of the package are used for those statements. If the package or routine contains dynamic SQL or XQuery statements, the authorization ID used for privilege checking depends on the setting of the DYNAMICRULES bind option of the package issuing the dynamic query statements, and whether those statements are issued when the package is being used in the context of a routine.

A user or group can be authorized for any combination of individual privileges or authorities. When a privilege is associated with an object, that object must exist. For example, a user cannot be given the SELECT privilege on a table unless that table has previously been created.

Note: Care must be taken when an authorization name representing a user or group is granted authorities and privileges and there is no user or group created with that name. At some later time, a user or group can be created with that name and automatically receive all of the authorities and privileges associated with that authorization name.

The REVOKE statement is used to revoke previously granted privileges. The revoking of a privilege from an authorization name revokes the privilege granted by all authorization names.

Revoking a privilege from an authorization name does not revoke that same privilege from any other authorization names that were granted the privilege by that authorization name. For example, assume that CLAIRE grants SELECT WITH GRANT OPTION to RICK, then RICK grants SELECT to BOBBY and CHRIS. If CLAIRE revokes the SELECT privilege from RICK, BOBBY and CHRIS still retain the SELECT privilege.

Object ownership:

When an object is created, one authorization ID is assigned *ownership* of the object. Ownership means the user is authorized to reference the object in any applicable SQL or XQuery statement.

When an object is created within a schema, the authorization ID of the statement must have the required privilege to create objects in the implicitly or explicitly specified schema. That is, the authorization name must either be the owner of the schema, or possess the CREATEIN privilege on the schema.

Note: This requirement is not applicable when creating table spaces, buffer pools or database partition groups. These objects are not created in schemas.

When an object is created, the authorization ID of the statement is the owner of that object.

Note: One exception exists. If the AUTHORIZATION option is specified for the CREATE SCHEMA statement, any other object that is created as part of the CREATE SCHEMA operation is owned by the authorization ID specified by the AUTHORIZATION option. Any objects that are created in the schema after the initial CREATE SCHEMA operation, however, are owned by the authorization ID associated with the specific CREATE statement.

For example, the statement `CREATE SCHEMA SCOTTSTUFF AUTHORIZATION SCOTT CREATE TABLE T1 (C1 INT)` creates the schema `SCOTTSTUFF` and the table `SCOTTSTUFF.T1`, which are both owned by SCOTT. Assume that the user BOBBY is granted the CREATEIN privilege on the `SCOTTSTUFF` schema and creates an index on the `SCOTTSTUFF.T1` table. Because the index is created after the schema, BOBBY owns the index on `SCOTTSTUFF.T1`.

Privileges are assigned to the object owner based on the type of object being created:

- The CONTROL privilege is implicitly granted on newly created tables, indexes, and packages. This privilege allows the object creator to access the database object, and to grant and revoke privileges to or from other users on that object. If a different user requires the CONTROL privilege to that object, a user with SYSADM or DBADM authority must grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked by the object owner.
- The CONTROL privilege is implicitly granted on newly created views if the object owner has the CONTROL privilege on all the tables, views, and nicknames referenced by the view definition.
- Other objects like triggers, routines, sequences, table spaces, and buffer pools do not have a CONTROL privilege associated with them. The object owner does, however, automatically receive each of the privileges associated with the object (and can provide these privileges to other users, where supported, by using the WITH GRANT option of the GRANT statement). In addition, the object owner can alter, add a comment on, or drop the object. These authorizations are implicit for the object owner and cannot be revoked.

Certain privileges on the object, such as altering a table, can be granted by the owner, and can be revoked from the owner by a user who has SYSADM or DBADM authority. Certain privileges on the object, such as commenting on a table, cannot be granted by the owner and cannot be revoked from the owner. Use the TRANSFER OWNERSHIP statement to move these privileges to another user. When an object is created, the authorization ID of the statement is the owner of the object. However, when a package is created and the OWNER bind option is specified, the owner of objects created by the static SQL statements in the package is the value of the OWNER bind option. In addition, if the AUTHORIZATION clause is specified on a CREATE SCHEMA statement, the authorization name specified after the AUTHORIZATION keyword is the owner of the schema.

A security administrator (SECADM) or the object owner can use the TRANSFER OWNERSHIP statement to change the ownership of a database object. An administrator can therefore create an object on behalf of an authorization ID, by creating the object using the authorization ID as the qualifier, and then using the TRANSFER OWNERSHIP statement to transfer the ownership that the administrator has on the object to the authorization ID.

Related concepts:

- "Controlling access to database objects" on page 95
- "Database administration authority (DBADM)" on page 76
- "Database authorities" on page 74
- "Index privileges" on page 82
- "Indirect privileges through a package" on page 99
- "LOAD authority" on page 77
- "Package privileges" on page 81
- "Routine privileges" on page 83
- "Schema privileges" on page 78
- "Security administration authority (SECADM)" on page 75
- "Sequence privileges" on page 82
- "System administration authority (SYSADM)" on page 71
- "System control authority (SYSCTRL)" on page 72
- "System maintenance authority (SYSMAINT)" on page 72
- "System monitor authority (SYSMON)" on page 73
- "Table and view privileges" on page 79
- "Table space privileges" on page 79

Related reference:

- "GRANT (Database Authorities) " on page 1081

Object creation, ownership, and privileges

When an object is created, one authorization name is assigned *ownership* of the object. Ownership means that the user is authorized to reference the object in any SQL or XQuery statement.

When an object is created within a schema, the authorization ID of the statement must have the required privilege to create objects in the implicitly or explicitly specified schema. That is, the authorization name must either be the owner of the schema, or possess the CREATEIN privilege on the schema.

Note: This requirement is not applicable when creating table spaces, buffer pools or database partition groups. These objects are not created in schemas.

When an object is created, the authorization ID of the statement is the owner of that object.

Note: One exception exists. If the AUTHORIZATION option is specified for the CREATE SCHEMA statement, any other object that is created as part of the CREATE SCHEMA operation is owned by the authorization ID specified by the AUTHORIZATION option. Any objects that are created in the schema after the initial CREATE SCHEMA operation, however, are owned by the authorization ID associated with the specific CREATE statement.

For example, the statement `CREATE SCHEMA SCOTTSTUFF AUTHORIZATION SCOTT CREATE TABLE T1 (C! INT)` creates the schema `SCOTTSTUFF` and the table `SCOTTSTUFF.T1`, which are both owned by SCOTT. Assume that the user BOBBY is granted the CREATEIN privilege on the `SCOTTSTUFF` schema and creates an index on the `SCOTTSTUFF.T1` table. Because the index is created after the schema, BOBBY owns the index on `SCOTTSTUFF.T1`.

Privileges are assigned to the object owner based on the type of object being created:

- The CONTROL privilege is implicitly granted on newly created tables, indexes, and packages. This privilege allows the object creator to access the database object, and to grant and revoke privileges to or from other users on that object. If a different user requires the CONTROL privilege to that object, a user with SYSADM or DBADM authority must grant the CONTROL privilege to that object. The CONTROL privilege cannot be revoked by the object owner.
- The CONTROL privilege is implicitly granted on newly created views if the object owner has the CONTROL privilege on all the tables, views, and nicknames referenced by the view definition.
- Other objects like triggers, routines, sequences, table spaces, and buffer pools do not have a CONTROL privilege associated with them. The object owner does, however, automatically receive each of the privileges associated with the object (and can provide these privileges to other users, where supported, by using the WITH GRANT option of the GRANT statement). In addition, the object owner can alter, add a comment on, or drop the object. These authorizations are implicit for the object owner and cannot be revoked.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Schemas

A *schema* is a collection of named objects; it provides a way to group those objects logically. A schema is also a name qualifier; it provides a way to use the same natural name for several objects, and to prevent ambiguous references to those objects; for example, the schema names 'INTERNAL' and 'EXTERNAL' make it easy to distinguish two different SALES tables (INTERNAL.SALES, EXTERNAL.SALES).

Schemas also enable multiple applications to store data in a single database without encountering namespace collisions.

A schema is distinct from, and should not be confused with, an *XML schema*, which is a standard that describes the structure and validates the content of XML documents.

A schema can contain tables, views, nicknames, triggers, functions, packages, and other objects. A schema is itself a database object. It is explicitly created using the CREATE SCHEMA statement, with the current user or a specified authorization ID recorded as the schema owner. It can also be implicitly created when another object is created, if the user has IMPLICIT_SCHEMA database authority.

A *schema name* is used as the high order part of a two-part object name. If the object is specifically qualified with a schema name when created, the object is assigned to that schema. If no schema name is specified when the object is created, the default schema name is used.

For example, a user with DBADM authority creates a schema called C for user A:

```
CREATE SCHEMA C AUTHORIZATION A
```

User A can then issue the following statement to create a table called X in schema C (provided that user A has the CREATETAB database authority):

```
CREATE TABLE C.X (COL1 INT)
```

Some schema names are reserved. For example, built-in functions belong to the SYSIBM schema, and the pre-installed user-defined functions belong to the SYSFUN schema.

When a database is created, all users have IMPLICIT_SCHEMA authority. This allows any user to create objects in any schema that does not already exist. An implicitly-created schema allows any user to create other objects in this schema. The ability to create aliases, distinct types, functions, and triggers is extended to implicitly-created schemas. The default privileges on an implicitly-created schema provide backward compatibility with previous versions.

If IMPLICIT_SCHEMA authority is revoked from PUBLIC, schemas can be explicitly created using the CREATE SCHEMA statement, or implicitly created by users (such as those with DBADM authority) who have been granted IMPLICIT_SCHEMA authority. Although revoking IMPLICIT_SCHEMA authority from PUBLIC increases control over the use of schema names, it can result in authorization errors when existing applications attempt to create objects.

Schemas also have privileges, allowing the schema owner to control which users have the privilege to create, alter, copy, and drop objects in the schema. This provides a way to control the manipulation of a subset of objects in the database. A schema owner is initially given all of these privileges on the schema, with the ability to grant the privileges to others. An implicitly-created schema is owned by the system, and all users are initially given the privilege to create objects in such a schema. A user with SYSADM or DBADM authority can change the privileges that are held by users on any schema. Therefore, access to create, alter, copy, and drop objects in any schema (even one that was implicitly created) can be controlled.

Related concepts:

- "Grouping objects by schema" on page 277

- "Schema privileges" on page 78

Related tasks:

- "Creating a schema" on page 304

Related reference:

- "Reserved schema names and reserved words" on page 29

Details on privileges, authorities, and authorization

Each authority is discussed in this section followed by the different privileges.

System administration authority (SYSADM)

The SYSADM authority level is the highest level of administrative authority. Users with SYSADM authority can run utilities, issue database and database manager commands, and access any data that is not protected by LBAC in any table in any database within the database manager instance. It provides the ability to control all database objects in the instance, including databases, tables, views, indexes, packages, schemas, servers, aliases, data types, functions, procedures, triggers, table spaces, database partition groups, buffer pools, and event monitors.

SYSADM authority is assigned to the group specified by the *sysadm_group* configuration parameter. Membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSADM authority can perform the following functions:

- Migrate a database
- Change the database manager configuration file (including specifying the groups having SYSCTRL, SYSMAINT, or SYSMON authority)
- Grant and revoke DBADM authority.
- Grant and revoke SECADM authority

While SYSADM authority does provide all abilities provided by most other authorities, it does not provide any of the abilities of the SECADM authority. The abilities provided by the SECADM authority are not provided by any other authority. SYSADM authority also does not provide access to data that is protected by LBAC.

Note: When a user with SYSADM authority creates a database, that user is automatically granted explicit DBADM authority on the database. If the database creator is removed from the SYSADM group and you want to prevent that user from accessing that database as a DBADM, you must explicitly revoke the user's DBADM authority.

Related concepts:

- "Data encryption" on page 104
- "Security administration authority (SECADM)" on page 75
- "System control authority (SYSCTRL)" on page 72
- "System maintenance authority (SYSMAINT)" on page 72
- "System monitor authority (SYSMON)" on page 73

System control authority (SYSCTRL)

SYSCTRL authority is the highest level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases. System control authority is designed for users administering a database manager instance containing sensitive data.

SYSCTRL authority is assigned to the group specified by the *sysctrl_group* configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSCTRL authority or higher can do the following:
- Update a database, node, or distributed connection services (DCS) directory
- Force users off the system
- Create or drop a database
- Drop, create, or alter a table space
- Restore to a new database.

In addition, a user with SYSCTRL authority can perform the functions of users with system maintenance authority (SYSMAINT) and system monitor authority (SYSMON).

Users with SYSCTRL authority also have the implicit privilege to connect to a database.

Note: When users with SYSCTRL authority create databases, they are automatically granted explicit DBADM authority on the database. If the database creator is removed from the SYSCTRL group, and if you want to also prevent them from accessing that database as a DBADM, you must explicitly revoke this DBADM authority.

Related concepts:
- "Database administration authority (DBADM)" on page 76
- "System maintenance authority (SYSMAINT)" on page 72
- "System monitor authority (SYSMON)" on page 73

System maintenance authority (SYSMAINT)

SYSMAINT authority is the second level of system control authority. This authority provides the ability to perform maintenance and utility operations against the database manager instance and its databases. These operations can affect system resources, but they do not allow direct access to data in the databases. System maintenance authority is designed for users maintaining databases within a database manager instance that contains sensitive data.

SYSMAINT authority is assigned to the group specified by the *sysmaint_group* configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

Only a user with SYSMAINT or higher system authority can do the following:
- Update database configuration files
- Back up a database or table space
- Restore to an existing database
- Perform roll forward recovery
- Start or stop an instance
- Restore a table space
- Run trace
- Take database system monitor snapshots of a database manager instance or its databases.

A user with SYSMAINT, DBADM, or higher authority can do the following:
- Query the state of a table space
- Update log history files
- Quiesce a table space
- Reorganize a table
- Collect catalog statistics using the `RUNSTATS` utility.

Users with SYSMAINT authority also have the implicit privilege to connect to a database, and can perform the functions of users with system monitor authority (SYSMON).

Related concepts:
- "Database administration authority (DBADM)" on page 76
- "System monitor authority (SYSMON)" on page 73

System monitor authority (SYSMON)

SYSMON authority provides the ability to take database system monitor snapshots of a database manager instance or its databases. SYSMON authority is assigned to the group specified by the *sysmon_group* configuration parameter. If a group is specified, membership in that group is controlled outside the database manager through the security facility used on your platform.

SYSMON authority enables the user to run the following commands:
- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

SYSMON authority enables the user to use the following APIs:
- db2GetSnapshot - Get Snapshot
- db2GetSnapshotSize - Estimate Size Required for db2GetSnapshot() Output Buffer
- db2MonitorSwitches - Get/Update Monitor Switches
- db2ResetMonitor - Reset Monitor

SYSMON authority enables the user use the following SQL table functions:

- All snapshot table functions without previously running `SYSPROC.SNAP_WRITE_FILE`

  `SYSPROC.SNAP_WRITE_FILE` takes a snapshot and saves its content into a file. If any snapshot table functions are called with null input parameters, the file content is returned instead of a real-time system snapshot.

Users with the SYSADM, SYSCTRL, or SYSMAINT authority level also possess SYSMON authority.

Related reference:

- "sysmon_group - System monitor authority group name " on page 1504

Database authorities

Each database authority allows the authorization ID holding it to perform some particular type of action on the database as a whole. Database authorities are different from privileges, which allow a certain action to be taken on a particular database object, such as a table or an index. There are ten different database authorities.

SECADM
Gives the holder the ability to configure many things related to security of the database, and also to transfer ownership of database objects. For instance, all objects that are part of the label-based access control (LBAC) feature can be created, dropped, granted, or revoked by a user that holds SECADM authority. SECADM specific abilities cannot be exercised by any other authority, not even SYSADM.

DBADM
Gives the holder the authority to act as the database administrator. In particular it gives the holder all of the other database authorities except for SECADM.

CONNECT
Allows the holder to connect to the database.

BINDADD
Allows the holder to create new packages in the database.

CREATETAB
Allows the holder to create new tables in the database.

CREATE_EXTERNAL_ROUTINE
Allows the holder to create a procedure for use by applications and other users of the database.

CREATE_NOT_FENCED_ROUTINE
Allows the holder to create a user-defined function (UDF) or procedure that is "not fenced". CREATE_EXTERNAL_ROUTINE is automatically granted to any user who is granted CREATE_NOT_FENCED_ROUTINE.

Attention:: The database manager does not protect its storage or control blocks from UDFs or procedures that are "not fenced". A user with this authority must, therefore, be very careful to test their UDF extremely well before registering it as "not fenced".

IMPLICIT_SCHEMA
Allows any user to create a schema implicitly by creating an object using a CREATE statement with a schema name that does not already exist.

SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

LOAD
Allows the holder to load data into a table

QUIESCE_CONNECT
Allows the holder to access the database while it is quiesced.

Only authorization IDs with the SYSADM authority can grant the SECADM and DBADM authorities. All other authorities can be granted by authorization IDs that hold SYSADM or DBADM authorities.

When a database is created, the following database authorities are automatically granted to PUBLIC for the new database:
- CREATETAB
- BINDADD
- CONNECT
- IMPLICIT_SCHEMA

In addition, these privileges are granted:
- USE privilege on USERSPACE1 table space
- SELECT privilege on the system catalog views.

To remove any database authority from PUBLIC, an authorization ID with DBADM or SYSADM authority must explicitly revoke it.

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Security administration authority (SECADM)

SECADM authority can only be granted by the SYSADM and can be granted to a user but not to a group. It gives these and only these abilities:
- Create and drop security label components
- Create and drop security policies
- Create and drop security labels
- Grant and revoke security labels
- Grant and revoke LBAC rule exemptions
- Grant and revoke setsessionuser privileges
- Execute the SQL statement TRANSFER OWNERSHIP on objects that you do not own

No other authority gives these abilities, not even SYSADM.

Related concepts:
- "Database authorities" on page 74
- "System administration authority (SYSADM)" on page 71
- "System control authority (SYSCTRL)" on page 72
- "System maintenance authority (SYSMAINT)" on page 72

Database administration authority (DBADM)

DBADM authority is an administrative authority for a specific database and it allows the user to perform certain actions, and issue database commands on that database. The DBADM authority allows access to the data in any table in the database unless that data is protected by LBAC. To access data protected by LBAC you must have appropriate LBAC credentials.

When DBADM authority is granted, the following database authorities are also explicitly granted for the same database (and are not automatically revoked if the DBADM authority is later revoked):

- BINDADD
- CONNECT
- CREATETAB
- CREATE_EXTERNAL
- ROUTINE, CREATE
- NOT_FENCED_ROUTINE
- IMPLICIT_SCHEMA
- QUIESCE_CONNECT
- LOAD

Only a user with SYSADM authority can grant or revoke DBADM authority. Users with DBADM authority can grant privileges on the database to others and can revoke any privilege from any user regardless of who granted it.

Holding the DBADM, or higher, authority for a database allows a user to perform these actions on that database:

- Read log files
- Create, activate, and drop event monitors.

A user with DBADM authority for a database or with SYSMAINT authority or higher can perform these actions on the database:

- Query the state of a table space
- Update log history files
- Quiesce a table space.
- Reorganize a table
- Collect catalog statistics using the `RUNSTATS` utility.

While DBADM authority does provide some of the same abilities as other authorities, it does not provide any of the abilities of the SECADM authority. The abilities provided by the SECADM authority are not provided by any other authority.

Related concepts:

- “Database authorities” on page 74
- “Implicit schema authority (IMPLICIT_SCHEMA) considerations” on page 77
- “LOAD authority” on page 77
- “Security administration authority (SECADM)” on page 75
- “System administration authority (SYSADM)” on page 71
- “System control authority (SYSCTRL)” on page 72

- "System maintenance authority (SYSMAINT)" on page 72

LOAD authority

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can use the `LOAD` command to load data into a table.

Users having LOAD authority at the database level, as well as INSERT privilege on a table, can `LOAD RESTART` or `LOAD TERMINATE` if the previous load operation is a load to insert data.

Users having LOAD authority at the database level, as well as the INSERT and DELETE privileges on a table, can use the `LOAD REPLACE` command.

If the previous load operation was a load replace, the DELETE privilege must also have been granted to that user before the user can `LOAD RESTART` or `LOAD TERMINATE`.

If the exception tables are used as part of a load operation, the user must have INSERT privilege on the exception tables.

The user with this authority can perform `QUIESCE TABLESPACES FOR TABLE`, `RUNSTATS`, and `LIST TABLESPACES` commands.

Related concepts:
- "Table and view privileges" on page 79
- "Privileges, authorities, and authorizations required to use Load" on page 414

Related reference:
- "LIST TABLESPACES " on page 537
- "LOAD " on page 542
- "QUIESCE TABLESPACES FOR TABLE " on page 569
- "RUNSTATS command" in *Command Reference*

Implicit schema authority (IMPLICIT_SCHEMA) considerations

When a new database is created, PUBLIC is given IMPLICIT_SCHEMA database authority. With this authority, any user can create a schema by creating an object and specifying a schema name that does not already exist. SYSIBM becomes the owner of the implicitly created schema and PUBLIC is given the privilege to create objects in this schema.

If control of who can implicitly create schema objects is required for the database, IMPLICIT_SCHEMA database authority should be revoked from PUBLIC. Once this is done, there are only three (3) ways that a schema object is created:
- Any user can create a schema using their own authorization name on a CREATE SCHEMA statement.
- Any user with DBADM authority can explicitly create any schema which does not already exist, and can optionally specify another user as the owner of the schema.
- Any user with DBADM authority has IMPLICIT_SCHEMA database authority (independent of PUBLIC) so that they can implicitly create a schema with any

name at the time they are creating other database objects. SYSIBM becomes the owner of the implicitly created schema and PUBLIC has the privilege to create objects in the schema.

Related tasks:

- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Schema privileges

Schema privileges are in the object privilege category. Object privileges are shown in Figure 17.

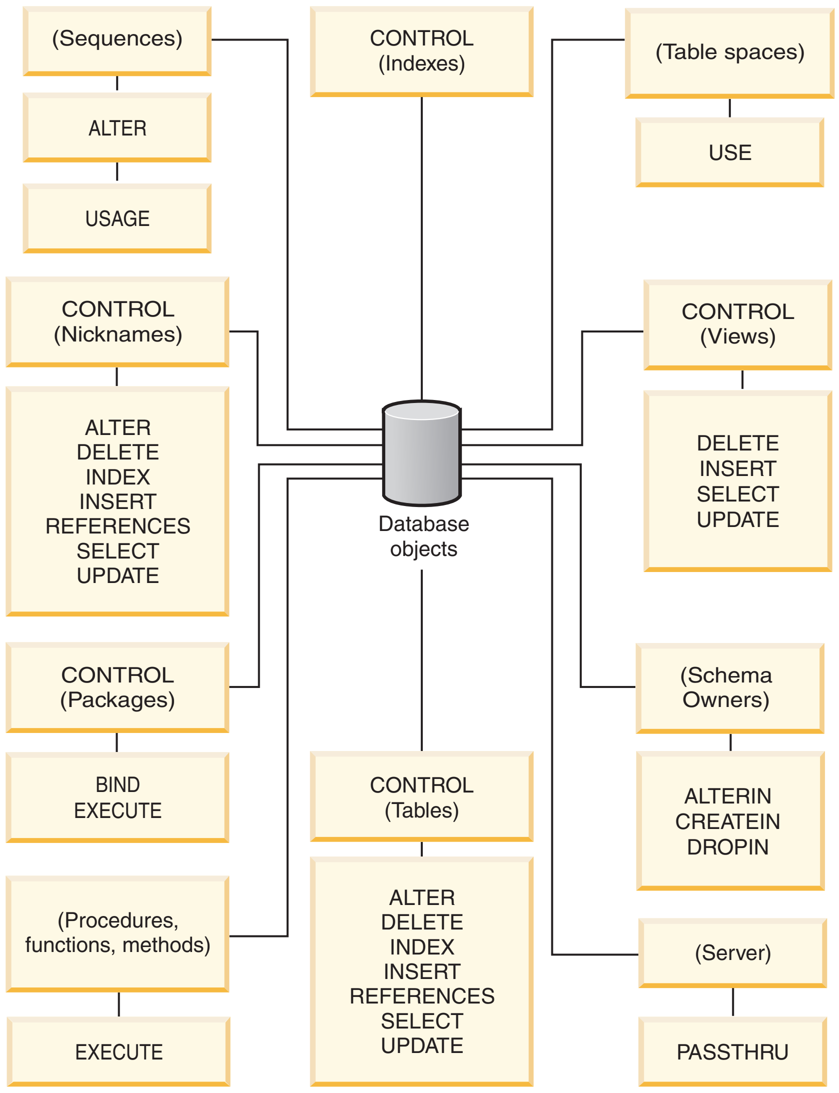


Figure 17. Object Privileges

Schema privileges involve actions on schemas in a database. A user may be granted any of the following privileges:

- CREATEIN allows the user to create objects within the schema.
- ALTERIN allows the user to alter objects within the schema.

- DROPIN allows the user to drop objects from within the schema.

The owner of the schema has all of these privileges and the ability to grant them to others. The objects that are manipulated within the schema object include: tables, views, indexes, packages, data types, functions, triggers, procedures, and aliases.

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Related reference:
- "ALTER SEQUENCE statement" in *SQL Reference, Volume 2*

Table space privileges

The table space privileges involve actions on the table spaces in a database. A user may be granted the USE privilege for a table space which then allows them to create tables within the table space.

The owner of the table space, typically the creator who has SYSADM or SYSCTRL authority, has the USE privilege and the ability to grant this privilege to others. By default, at database creation time the USE privilege for table space USERSPACE1 is granted to PUBLIC, though this privilege can be revoked.

The USE privilege cannot be used with SYSCATSPACE or any system temporary table spaces.

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Related reference:
- "CREATE TABLE " on page 956

Table and view privileges

Table and view privileges involve actions on tables or views in a database. A user must have CONNECT authority on the database to use any of the following privileges:
- CONTROL provides the user with all privileges for a table or view including the ability to drop it, and to grant and revoke individual table privileges. You must have SYSADM or DBADM authority to grant CONTROL. The creator of a table automatically receives CONTROL privilege on the table. The creator of a view automatically receives CONTROL privilege only if they have CONTROL privilege on all tables, views, and nicknames referenced in the view definition, or they have SYSADM or DBADM authority.
- ALTER allows the user to modify on a table, for example, to add columns or a unique constraint to the table. A user with ALTER privilege can also COMMENT ON a table, or on columns of the table. For information about the possible modifications that can be performed on a table, see the ALTER TABLE and COMMENT statements.
- DELETE allows the user to delete rows from a table or view.

- INDEX allows the user to create an index on a table. Creators of indexes automatically have CONTROL privilege on the index.
- INSERT allows the user to insert a row into a table or view, and to run the IMPORT utility.
- REFERENCES allows the user to create and drop a foreign key, specifying the table as the parent in a relationship. The user might have this privilege only on specific columns.
- SELECT allows the user to retrieve rows from a table or view, to create a view on a table, and to run the EXPORT utility.
- UPDATE allows the user to change an entry in a table, a view, or for one or more specific columns in a table or view. The user may have this privilege only on specific columns.

The privilege to grant these privileges to others may also be granted using the WITH GRANT OPTION on the GRANT statement.

Note: When a user or group is granted CONTROL privilege on a table, all other privileges on that table are automatically granted WITH GRANT OPTION. If you subsequently revoke the CONTROL privilege on the table from a user, that user will still retain the other privileges that were automatically granted. To revoke all the privileges that are granted with the CONTROL privilege, you must either explicitly revoke each individual privilege or specify the ALL keyword on the REVOKE statement, for example:

```
REVOKE ALL
  ON EMPLOYEE FROM USER HERON
```

When working with typed tables, there are implications regarding table and view privileges.

Note: Privileges may be granted independently at every level of a table hierarchy. As a result, a user granted a privilege on a supertable within a hierarchy of typed tables may also indirectly affect any subtables. However, a user can only operate directly on a subtable if the necessary privilege is held on that subtable.

The supertable/subtable relationships among the tables in a table hierarchy mean that operations such as SELECT, UPDATE, and DELETE will affect the rows of the operation's target table and all its subtables (if any). This behavior can be called *substitutability*. For example, suppose that you have created an Employee table of type Employee_t with a subtable Manager of type Manager_t. A manager is a (specialized) kind of employee, as indicated by the type/subtype relationship between the structured types Employee_t and Manager_t and the corresponding table/subtable relationship between the tables Employee and Manager. As a result of this relationship, the SQL query:

```
SELECT * FROM Employee
```

will return the object identifier and Employee_t attributes for both employees and managers. Similarly, the update operation:

```
UPDATE Employee SET Salary = Salary + 1000
```

will give a thousand dollar raise to managers as well as regular employees.

A user with SELECT privilege on Employee will be able to perform this SELECT operation even if they do not have an explicit SELECT privilege on Manager. However, such a user will not be permitted to perform a SELECT operation directly on the Manager subtable, and will therefore not be able to access any of the non-inherited columns of the Manager table.

Similarly, a user with UPDATE privilege on Employee will be able to perform an UPDATE operation on Manager, thereby affecting both regular employees and managers, even without having the explicit UPDATE privilege on the Manager table. However, such a user will not be permitted to perform UPDATE operations directly on the Manager subtable, and will therefore not be able to update non-inherited columns of the Manager table.

Related concepts:
- "Index privileges" on page 82

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Related reference:
- "ALTER TABLE " on page 854
- "CREATE VIEW " on page 1034
- "SELECT " on page 1182

Package privileges

A package is a database object that contains the information needed by the database manager to access data in the most efficient way for a particular application program. Package privileges enable a user to create and manipulate packages. The user must have CONNECT authority on the database to use any of the following privileges:
- CONTROL provides the user with the ability to rebind, drop, or execute a package as well as the ability to extend those privileges to others. The creator of a package automatically receives this privilege. A user with CONTROL privilege is granted the BIND and EXECUTE privileges, and can also grant these privileges to other users by using the GRANT statement. (If a privilege is granted using WITH GRANT OPTION, a user who receives the BIND or EXECUTE privilege can, in turn, grant this privilege to other users.) To grant CONTROL privilege, the user must have SYSADM or DBADM authority.
- BIND privilege on a package allows the user to rebind or bind that package and to add new package versions of the same package name and creator.
- EXECUTE allows the user to execute or run a package.

Note: All package privileges apply to all VERSIONs that share the same package name and creator.

In addition to these package privileges, the BINDADD database privilege allows users to create new packages or rebind an existing package in the database.

Objects referenced by nicknames need to pass authentication checks at the data sources containing the objects. In addition, package users must have the appropriate privileges or authority levels for data source objects at the data source.

It is possible that packages containing nicknames might require additional authorization steps because DB2 database uses dynamic queries when communicating with DB2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source.

Related concepts:
- "Database authorities" on page 74

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Index privileges

The creator of an index or an index specification automatically receives CONTROL privilege on the index. CONTROL privilege on an index is really the ability to drop the index. To grant CONTROL privilege on an index, a user must have SYSADM or DBADM authority.

The table-level INDEX privilege allows a user to create an index on that table.

The nickname-level INDEX privilege allows a user to create an index specification on that nickname.

Related concepts:
- "Table and view privileges" on page 79

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Sequence privileges

The creator of a sequence automatically receives the USAGE and ALTER privileges on the sequence. The USAGE privilege is needed to use NEXT VALUE and PREVIOUS VALUE expressions for the sequence. To allow other users to use the NEXT VALUE and PREVIOUS VALUE expressions, sequence privileges must be granted to PUBLIC. This allows all users to use the expressions with the specified sequence.

ALTER privilege on the sequence allows the user to perform tasks such as restarting the sequence or changing the increment for future sequence values. The creator of the sequence can grant the ALTER privilege to other users, and if WITH GRANT OPTION is used, these users can, in turn, grant these privileges to other users.

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Related reference:
- "ALTER SEQUENCE statement" in *SQL Reference, Volume 2*

Routine privileges

Execute privileges involve actions on all types of routines such as functions, procedures, and methods within a database. Once having EXECUTE privilege, a user can then invoke that routine, create a function that is sourced from that routine (applies to functions only), and reference the routine in any DDL statement such as CREATE VIEW or CREATE TRIGGER.

The user who defines the externally stored procedure, function, or method receives EXECUTE WITH GRANT privilege. If the EXECUTE privilege is granted to another user via WITH GRANT OPTION, that user can, in turn, grant the EXECUTE privilege to another user.

Related tasks:
- “Granting privileges” on page 95
- “Revoking privileges” on page 97

Authorizations and binding of routines that contain SQL

When discussing routine level authorization it is important to define some roles related to routines, the determination of the roles, and the privileges related to these roles:

Package Owner
The owner of a particular package that participates in the implementation of a routine. The package owner is the user who executes the BIND command to bind a package with a database, unless the OWNER precompile/BIND option is used to override the package ownership and set it to an alternate user. Upon execution of the BIND command, the package owner is granted EXECUTE WITH GRANT privilege on the package. A routine library or executable can be comprised of multiple packages and therefore can have multiple package owners associated with it.

Routine Definer

The ID that issues the CREATE statement to register a routine. The routine definer is generally a DBA, but is also often the routine package owner. When a routine is invoked, at package load time, the authorization to run the routine is checked against the definer’s authorization to execute the package or packages associated with the routine (not against the authorization of the routine invoker). For a routine to be successfully invoked, the routine definer must have one of:
- EXECUTE privilege on the package or packages of the routine and EXECUTE privilege on the routine
- SYSADM or DBADM authority

If the routine definer and the routine package owner are the same user, then the routine definer will have the required EXECUTE privileges on the packages. If the definer is not the package owner, the definer must be explicitly granted EXECUTE privilege on the packages by the package owner or any user with SYSADM or DBADM authority.

Upon issuing the CREATE statement that registers the routine, the definer is implicitly granted the EXECUTE WITH GRANT OPTION privilege on the routine.

The routine definer's role is to encapsulate under one authorization ID, the privileges of running the packages associated with a routine and the privilege of granting EXECUTE privilege on the routine to PUBLIC or to specific users that need to invoke the routine.

Note: For SQL routines the routine definer is also implicitly the package owner. Therefore the definer will have EXECUTE WITH GRANT OPTION on both the routine and on the routine package upon execution of the CREATE statement for the routine.

Routine Invoker
The ID that invokes the routine. To determine which users will be invokers of a routine, it is necessary to consider how a routine can be invoked. Routines can be invoked from a command window or from within an embedded SQL application. In the case of methods and UDFs the routine reference will be embedded in another SQL statement. A procedure is invoked by using the CALL statement. For dynamic SQL in an application, the invoker is the runtime authorization ID of the immediately higher-level routine or application containing the routine invocation (however, this ID can also depend on the DYNAMICRULES option with which the higher-level routine or application was bound). For static SQL, the invoker is the value of the OWNER precompile/BIND option of the package that contains the reference to the routine. To successfully invoke the routine, these users will require EXECUTE privilege on the routine. This privilege can be granted by any user with EXECUTE WITH GRANT OPTION privilege on the routine (this includes the routine definer unless the privilege has been explicitly revoked), SYSADM or DBADM authority by explicitly issuing a GRANT statement.

As an example, if a package associated with an application containing dynamic SQL was bound with DYNAMICRULES BIND, then its runtime authorization ID will be its package owner, not the person invoking the package. Also, the package owner will be the actual binder or the value of the OWNER precompile/bind option. In this case, the invoker of the routine assumes this value rather than the ID of the user who is executing the application.

Notes:
1. For static SQL within a routine, the package owner's privileges must be sufficient to execute the SQL statements in the routine body. These SQL statements might require table access privileges or execute privileges if there are any nested references to routines.
2. For dynamic SQL within a routine, the userid whose privileges will be validated are governed by the DYNAMICRULES option of the BIND of the routine body.
3. The routine package owner must GRANT EXECUTE on the package to the routine definer. This can be done before or after the routine is registered, but it must be done before the routine is invoked otherwise an error (SQLSTATE 42051) will be returned.

The steps involved in managing the execute privilege on a routine are detailed in the diagram and text that follows:

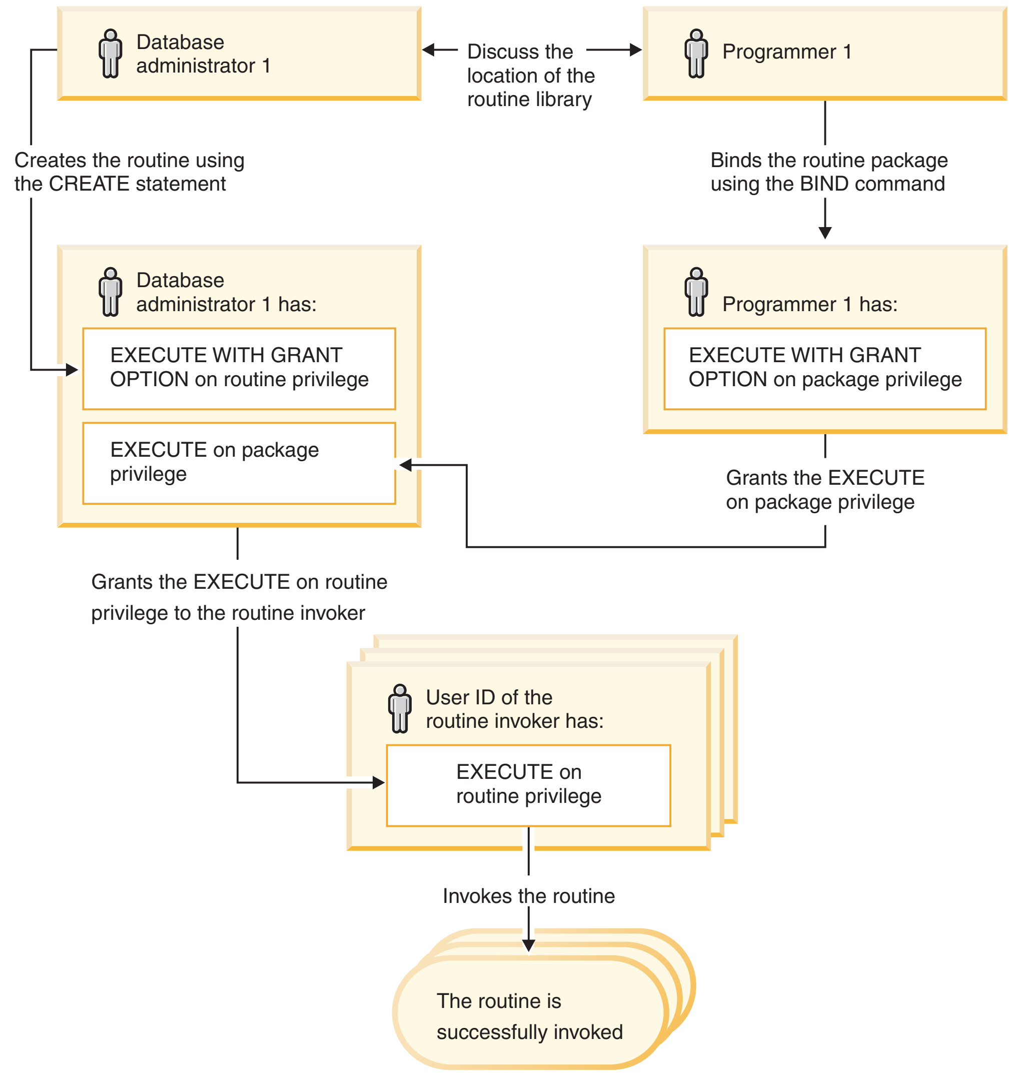

Database administrator 1
Discuss the location of the routine library
Programmer 1
Creates the routine using the CREATE statement
Binds the routine package using the BIND command
Database administrator 1 has:
EXECUTE WITH GRANT OPTION on routine privilege
EXECUTE on package privilege
Programmer 1 has:
EXECUTE WITH GRANT OPTION on package privilege
Grants the EXECUTE on package privilege
Grants the EXECUTE on routine privilege to the routine invoker
User ID of the routine invoker has:
EXECUTE on routine privilege
Invokes the routine
The routine is successfully invoked

Figure 18. Managing the EXECUTE privilege on routines

1. Definer performs the appropriate CREATE statement to register the routine. This registers the routine in DB2 with its intended level of SQL access, establishes the routine signature, and also points to the routine executable. The definer, if not also the package owner, needs to communicate with the package owners and authors of the routine programs to be clear on where the routine libraries reside so that this can be correctly specified in the EXTERNAL clause of the CREATE statement. By virtue of a successful CREATE statement, the definer has EXECUTE WITH GRANT privilege on the routine, however the definer does not yet have EXECUTE privilege on the packages of the routine.
2. Definer must grant EXECUTE privilege on the routine to any users who are to be permitted use of the routine. (If the package for this routine will recursively call this routine, then this step must be done before the next step.)
3. Package owners precompile and bind the routine program, or have it done on their behalf. Upon a successful precompile and bind, the package owner is implicitly granted EXECUTE WITH GRANT OPTION privilege on the respective package. This step follows step one in this list only to cover the possibility of SQL recursion in the routine. If such recursion does not exist in any particular case, the precompile/bind could precede the issuing of the CREATE statement for the routine.

4. Each package owner must explicitly grant EXECUTE privilege on their respective routine package to the definer of the routine. This step must come at some time after the previous step. If the package owner is also the routine definer, this step can be skipped.
5. Static usage of the routine: the bind owner of the package referencing the routine must have been given EXECUTE privilege on the routine, so the previous step must be completed at this point. When the routine executes, DB2 verifies that the definer has the EXECUTE privilege on any package that is needed, so step 3 must be completed for each such package.
6. Dynamic usage of the routine: the authorization ID as controlled by the DYNAMICRULES option for the invoking application must have EXECUTE privilege on the routine (step 4), and the definer of the routine must have the EXECUTE privilege on the packages (step 3).

Related concepts:
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 1308
- "Routine invocation" in *Developing SQL and External Routines*
- "Security of routines" on page 1292
- "Authorization, privileges, and object ownership" on page 63
- "Routine privileges" on page 83

Related reference:
- "BIND" on page 441
- "CREATE FUNCTION " on page 920
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*

Controlling Database Access

One of the most important responsibilities of the database administrator and the system administrator is database security. Securing your database involves several activities:
- Preventing accidental loss of data or data integrity through equipment or system malfunction.
- Preventing unauthorized access to valuable data. You must ensure that sensitive information is not accessed by those without a "need to know".
- Preventing unauthorized persons from committing mischief through malicious deletion or tampering with data.
- Monitoring access of data by users.

The following topics are discussed:
- "Security issues when installing the DB2 database manager" on page 87
- "Authentication methods for your server" on page 89
- "Authentication considerations for remote clients" on page 94
- "Introduction to firewall support" on page 157
- "Authorization, privileges, and object ownership" on page 63
- "Controlling access to database objects" on page 95
- "Tasks and required authorizations" on page 105
- "Using the system catalog for security issues" on page 161.

Planning for Security: Start by defining your objectives for a database access control plan, and specifying who shall have access to what and under what circumstances. Your plan should also describe how to meet these objectives by using database functions, functions of other programs, and administrative procedures.

Security issues when installing the DB2 database manager

Security considerations are important to the DB2 administrator from the moment the product is installed.

To complete the installation of the DB2 database manager, a user ID, a group name, and a password are required. The GUI-based DB2 database manager install program creates default values for different user IDs and the group. Different defaults are created, depending on whether you are installing on UNIX or Windows platforms:

- On UNIX and Linux platforms, if you choose to create a DB2 instance in the instance setup window, the DB2 database install program creates, by default, different users for the DAS (`dasusr`), the instance owner (`db2inst`), and the fenced user (`db2fenc`). Optionally, you can specify different user names

  The DB2 database install program appends a number from 1-99 to the default user name, until a user ID that does not already exist can be created. For example, if the users `db2inst1` and `db2inst2` already exist, the DB2 database install program creates the user `db2inst3`. If a number greater than `10` is used, the character portion of the name is truncated in the default user ID. For example, if the user ID `db2fenc9` already exists, the DB2 database install program truncates the `c` in the user ID, then appends the `10` (`db2fen10`). Truncation does not occur when the numeric value is appended to the default DAS user (for example, `dasusr24`).
- On Windows platforms, the DB2 database install program creates, by default, the user `db2admin` for the DAS user, the instance owner, and fenced users (you can specify a different user name during setup, if you want). Unlike UNIX platforms, no numeric value is appended to the user ID.

To minimize the risk of a user other than the administrator from learning of the defaults and using them in an improper fashion within databases and instances, change the defaults during the install to a new or existing user ID of your choice.

Note: Response file installations do not use default values for user IDs or group names. These values must be specified in the response file.

Passwords are very important when authenticating users. If no authentication requirements are set at the operating system level and the database is using the operating system to authenticate users, users will be allowed to connect. For example on UNIX operating systems, undefined passwords are treated as NULL. In this situation, any user without a defined password will be considered to have a NULL password. From the operating system's perspective, this is a match and the user is validated and able to connect to the database. Use passwords at the operating system level if you want the operating system to do the authentication of users for your database.

When working with DB2 Data Partitioning Feature (DPF) on UNIX operating system environments, the DB2 database manager by default uses the rsh utility (remsh on HP-UX) to run some commands on remote nodes. The rsh utility transmits passwords in clear text over the network, which can be a security exposure if the DB2 server is not on a secure network. You can use the DB2RSHCMD registry variable to set the remote shell program to a more secure alternative that avoids this exposure. One example of a more secure alternative is ssh. See the DB2RSHCMD registry variable documentation for restrictions on remote shell configurations.

After installing the DB2 database manager, also review, and change (if required), the default privileges that have been granted to users. By default, the installation process grants system administration (SYSADM) privileges to the following users on each operating system:

Windows environments	A valid DB2 database user name that belongs to the Administrators group.
UNIX platforms	A valid DB2 database user name that belongs to the primary group of the instance owner.

SYSADM privileges are the most powerful set of privileges available within the DB2 database manager. As a result, you might not want all of these users to have SYSADM privileges by default. The DB2 database manager provides the administrator with the ability to grant and revoke privileges to groups and individual user IDs.

By updating the database manager configuration parameter *sysadm_group*, the administrator can control which group of users possesses SYSADM privileges. You must follow the guidelines below to complete the security requirements for both the DB2 database installation and the subsequent instance and database creation.

Any group defined as the system administration group (by updating *sysadm_group*) must exist. The name of this group should allow for easy identification as the group created for instance owners. User IDs and groups that belong to this group have system administrator authority for their respective instances.

The administrator should consider creating an instance owner user ID that is easily recognized as being associated with a particular instance. This user ID should have as one of its groups the name of the SYSADM group created above. Another recommendation is to use this instance-owner user ID only as a member of the instance owner group and not to use it in any other group. This should control the proliferation of user IDs and groups that can modify the instance, or any object within the instance.

The created user ID must be associated with a password to provide authentication before being permitted entry into the data and databases within the instance. The recommendation when creating a password is to follow your organization's password naming guidelines.

Note: To avoid accidentally deleting or overwriting instance configuration or other files, administrators should consider using another user account, which does not belong to the same primary group as the instance owner, for day-to-day administration tasks that are performed on the server directly.

Related concepts:

- "General naming rules" on page 23
- "User, user ID and group naming rules" on page 26
- "Authentication" on page 61
- "Authorization" on page 62

- “Naming rules in a Unicode environment” on page 28
- “Naming rules in an NLS environment” on page 27
- “Location of the instance directory” in *Administration Guide: Implementation*
- “UNIX platform security considerations for users” on page 108
- “Windows platform security considerations for users” on page 107

Related reference:

- “Communications variables” in *Performance Guide*

Authentication methods for your server

Access to an instance or a database first requires that the user be *authenticated*. The *authentication type* for each instance determines how and where a user will be verified. The authentication type is stored in the database manager configuration file at the server. It is initially set when the instance is created. There is one authentication type per instance, which covers access to that database server and all the databases under its control.

If you intend to access data sources from a federated database, you must consider data source authentication processing and definitions for federated authentication types.

Note: You can check the following web site for certification information on the cryptographic routines used by the DB2 database management system to perform encryption of the userid and password when using SERVER_ENCRYPT authentication, and of the userid, password and user data when using DATA_ENCRYPT authentication: http://www.ibm.com/security/standards/st_evaluations.shtml.

The following authentication types are provided:

SERVER

Specifies that authentication occurs on the server using local operating system security. If a user ID and password are specified during the connection or attachment attempt, they are compared to the valid user ID and password combinations at the server to determine if the user is permitted to access the instance. This is the default security mechanism.

Notes:

1. The server code detects whether a connection is local or remote. For local connections, when authentication is SERVER, a user ID and password are not required for authentication to be successful.
2. If you are installing the DB2 database to set up a Common Criteria certified configuation, you must specify SERVER.

SERVER_ENCRYPT

Specifies that the server accepts encrypted SERVER authentication schemes. If the client authentication is not specified, the client is authenticated using the method selected at the server.

CLIENT

Specifies that authentication occurs on the database partition where the application is invoked using operating system security. The user ID and password specified during a connection or attachment attempt are compared with the valid user ID and password combinations on the client node to determine if the user ID is permitted access to the instance. No further authentication will take place on the database server. This is sometimes called single signon.

If the user performs a local or client login, the user is known only to that local client workstation.

If the remote instance has CLIENT authentication, two other parameters determine the final authentication type: *trust_allclnts* and *trust_clntauth*.

CLIENT level security for TRUSTED clients only:

Trusted clients are clients that have a reliable, local security system.

When the authentication type of CLIENT has been selected, an additional option may be selected to protect against clients whose operating environment has no inherent security.

To protect against unsecured clients, the administrator can select Trusted Client Authentication by setting the *trust_allclnts* parameter to NO. This implies that all trusted platforms can authenticate the user on behalf of the server. Untrusted clients are authenticated on the Server and must provide a user ID and password. You use the *trust_allclnts* configuration parameter to indicate whether you are trusting clients. The default for this parameter is YES.

Note: It is possible to trust all clients (*trust_allclnts* is YES) yet have some of those clients as those who do not have a native safe security system for authentication.

You may also want to complete authentication at the server even for trusted clients. To indicate where to validate trusted clients, you use the *trust_clntauth* configuration parameter. The default for this parameter is CLIENT.

Note: For trusted clients only, if no user ID or password is explicitly provided when attempting to CONNECT or ATTACH, then validation of the user takes place at the client. The *trust_clntauth* parameter is only used to determine where to validate the information provided on the USER or USING clauses.

To protect against all clients except DRDA® clients from DB2 for OS/390® and z/OS®, DB2 for VM and VSE, and DB2 for iSeries™, set the *trust_allclnts* parameter to DRDAONLY. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

The *trust_clntauth* parameter is used to determine where the above clients are authenticated: if *trust_clntauth* is "client", authentication takes place at the client. If *trust_clntauth* is "server", authentication takes place at the client when no user ID and password are provided and at the server when a user ID and password are provided.

Table 7. Authentication Modes using TRUST_ALLCLNTS and TRUST_CLNTAUTH Parameter Combinations.

TRUST_ ALLCLNTS	TRUST_ CLNTAUTH	Untrusted non– DRDA Client Authen- tication (no user ID & password)	Untrusted non– DRDA Client Authen- tication (with user ID & password)	Trusted non– DRDA Client Authen- tication (no user ID & password)	Trusted non– DRDA Client Authen- tication (with user ID & password)	DRDA Client Authen- tication (no user ID & password)	DRDA Client Authen- tication (with user ID & password)
YES	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT	CLIENT
YES	SERVER	CLIENT	SERVER	CLIENT	SERVER	CLIENT	SERVER
NO	CLIENT	SERVER	SERVER	CLIENT	CLIENT	CLIENT	CLIENT
NO	SERVER	SERVER	SERVER	CLIENT	SERVER	CLIENT	SERVER
DRDAONLY	CLIENT	SERVER	SERVER	SERVER	SERVER	CLIENT	CLIENT
DRDAONLY	SERVER	SERVER	SERVER	SERVER	SERVER	CLIENT	SERVER

KERBEROS

Used when both the DB2 client and server are on operating systems that support the Kerberos security protocol. The Kerberos security protocol performs authentication as a third party authentication service by using conventional cryptography to create a shared secret key. This key becomes a user's credential and is used to verify the identity of users during all occasions when local or network services are requested. The key eliminates the need to pass the user name and password across the network as clear text. Using the Kerberos security protocol enables the use of a single sign-on to a remote DB2 database server. The KERBEROS authentication type is supported on clients and servers running Windows, AIX, and Solaris operating environment.

Kerberos authentication works as follows:

1. A user logging on to the client machine using a domain account authenticates to the Kerberos key distribution center (KDC) at the domain controller. The key distribution center issues a ticket-granting ticket (TGT) to the client.
2. During the first phase of the connection the server sends the target principal name, which is the service account name for the DB2 database server service, to the client. Using the server's target principal name and the target-granting ticket, the client requests a service ticket from the ticket-granting service (TGS) which also resides at the domain controller. If both the client's ticket-granting ticket and the server's target principal name are valid, the TGS issues a service ticket to the client. The principal name recorded in the database directory may now be specified as name/instance@REALM. (This is in addition to the current DOMAIN\userID and userID@xxx.xxx.xxx.com formats accepted on Windows with DB2 UDB Version 7.1 and following.)
3. The client sends this service ticket to the server using the communication channel (which may be, as an example, TCP/IP).
4. The server validates the client's server ticket. If the client's service ticket is valid, then the authentication is completed.

It is possible to catalog the databases on the client machine and explicitly specify the Kerberos authentication type with the server's target principal name. In this way, the first phase of the connection can be bypassed.

If a user ID and a password are specified, the client will request the ticket-granting ticket for that user account and use it for authentication.

KRB_SERVER_ENCRYPT
Specifies that the server accepts KERBEROS authentication or encrypted SERVER authentication schemes. If the client authentication is KERBEROS, the client is authenticated using the Kerberos security system. If the client authentication is SERVER_ENCRYPT, the client is authenticated using a user ID and encryption password. If the client authentication is not specified, then the client will use Kerberos if available, otherwise it will use password encryption. For other client authentication types, an authentication error is returned. The authenticaion type of the client cannot be specified as KRB_SERVER_ENCRYPT

Note: The Kerberos authentication types are only supported on clients and servers running Windows, and AIX operating systems, as well as Solaris operating environment. Also, both client and server machines must either belong to the same Windows domain or belong to trusted domains. This authentication type should be used when the server supports Kerberos and some, but not all, of the client machines support Kerberos authentication.

DATA_ENCRYPT
The server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works exactly the same way as that shown with SERVER_ENCRYPT. See that authentication type for more information.

The following user data are encrypted when using this authentication type:
- SQL and XQuery statements.
- SQL program variable data.
- Output data from the server processing of an SQL or XQuery statement and including a description of the data.
- Some or all of the answer set data resulting from a query.
- Large object (LOB) data streaming.
- SQLDA descriptors.

DATA_ENCRYPT_CMP
The server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with down level products not supporting DATA_ENCRYPT authentication type. These products are permitted to connect with the SERVER_ENCRYPT authentication type and without encrypting user data. Products supporting the new authentication type must use it. This authentication type is only valid in the server's database manager configuration file and is not valid when used on on the CATALOG DATABASE command.

GSSPLUGIN
Specifies that the server uses a GSS-API plug-in to perform authentication. If the client authentication is not specified, the server returns a list of server-supported plug-ins, including any Kerberos plug-in that is listed in the *srvcon_gssplugin_list* database manager configuration parameter, to the client. The client selects the first plug-in found in the client plug-in directory from the list. If the client does not support any plug-in in the list, the client is authenticated using the Kerberos authentication scheme (if it is

returned). If the client authentication is the GSSPLUGIN authentication scheme, the client is authenticated using the first supported plug-in in the list.

GSS_SERVER_ENCRYPT
Specifies that the server accepts plug-in authentication or encrypted server authentication schemes. If client authentication occurs through a plug-in, the client is authenticated using the first client-supported plug-in in the list of server-supported plug-ins.

If the client authentication is not specified and an implicit connect is being performed (that is, the client does not supply a user ID and password when making the connection), the server returns a list of server-supported plug-ins, the Kerberos authentication scheme (if one of the plug-ins in the list is Kerberos-based), and the encrypted server authentication scheme. The client is authenticated using the first supported plug-in found in the client plug-in directory. If the client does not support any of the plug-ins that are in the list, the client is authenticated using the Kerberos authentication scheme. If the client does not support the Kerberos authentication scheme, the client is authenticated using the encrypted server authentication scheme, and the connection will fail because of a missing password. A client supports the Kerberos authentication scheme if a DB2-supplied Kerberos plug-in exists for the operating system, or a Kerberos-based plug-in is specified for the *srvcon_gssplugin_list* database manager configuration parameter.

If the client authentication is not specified and an explicit connection is being performed (that is, both the user ID and password are supplied), the authentication type is equivalent to SERVER_ENCRYPT.

Notes:

1. Do not inadvertently lock yourself out of your instance when you are changing the authentication information, since access to the configuration file itself is protected by information in the configuration file. The following database manager configuration file parameters control access to the instance:
   - AUTHENTICATION *
   - SYSADM_GROUP *
   - TRUST_ALLCLNTS
   - TRUST_CLNTAUTH
   - SYSCTRL_GROUP
   - SYSMAINT_GROUP

   * Indicates the two most important parameters, and those most likely to cause a problem.

   There are some things that can be done to ensure this does not happen: If you do accidentally lock yourself out of the DB2 database system, you have a fail-safe option available on all platforms that will allow you to override the usual DB2 database security checks to update the database manager configuration file using a highly privileged local operating system security user. This user *always* has the privilege to update the database manager configuration file and thereby correct the problem. However, this security bypass is restricted to a local update of the database manager configuration file. You cannot use a fail-safe user remotely or for any other DB2 database command. This special user is identified as follows:
   - UNIX platforms: the instance owner
   - Windows platform: someone belonging to the local “administrators” group

- Other platforms: there is no local security on the other platforms, so all users pass local security checks anyway

Related concepts:
- "Authentication considerations for remote clients" on page 94
- "DB2 and Windows security introduction" in *Administration Guide: Implementation*
- "Partitioned database authentication considerations" on page 62

Related reference:
- "authentication - Authentication type " on page 1497
- "trust_allclnts - Trust all clients " on page 1505
- "trust_clntauth - Trusted clients authentication " on page 1506

Authentication considerations for remote clients

When cataloging a database for remote access, the authentication type can be specified in the database directory entry.

The authentication type is not required. If it is not specified, the client will default to SERVER_ENCRYPT. However, if the server does not support SERVER_ENCRYPT, the client attempts to retry using a value supported by the server. If the server supports multiple authentication types, the client will not choose among them, but instead returns an error. The error is returned to ensure that the correct authentication type is used. In this case, the client must catalog the database using a supported authentication type. If an authentication type is specified, authentication can begin immediately provided that value specified matches that at the server. If a mismatch is detected, DB2 database attempts to recover. Recovery may result in more flows to reconcile the difference, or in an error if the DB2 database cannot recover. In the case of a mismatch, the value at the server is assumed to be correct.

The authentication type DATA_ENCRYPT_CMP is designed to allow clients from a previous release that does not support data encryption to a server using SERVER_ENCRYPT authentication instead of DATA_ENCRYPT. This authentication does not work when the following statements are true:
- The client level is Version 7.2.
- The gateway level is Version 8 FixPak7 or later.
- The server is Version 8 FixPak 7 or later.

When these are all true, the client cannot connect to the server. To allow the connection, you must either upgrade your client to Version 8, or have your gateway level at Version 8 FixPak 6 or earlier.

The determination of the authentication type used when connecting is made by specifying the appropriate authentication type as a database catalog entry at the gateway. This is true for both DB2 Connect™ scenarios and for clients and servers in a partitioned database environment where the client has set the DB2NODE registry variable. You will catalog the authentication type at the catalog partition with the intent to "hop" to the appropriate partition. In this scenario, the authentication type cataloged at the gateway is not used because the negotiation is solely between the client and the server.

You may have a need to catalog multiple database aliases at the gateway using different authentication types if they need to have clients that use differing authentication types. When deciding which authentication type to catalog at a gateway, you can keep the authentication type the same as that used at the client and server; or, you can use the NOTSPEC authentication type with the understanding that NOTSPEC defaults to SERVER.

Related concepts:
- "Authentication methods for your server" on page 89

Controlling access to database objects

Controlling data access requires an understanding of direct and indirect privileges, administrative authorities, and packages. This section explains these topics and provides some examples.

Directly granted privileges are stored in the system catalog.

Authorization is controlled in three ways:
- Explicit authorization is controlled through privileges controlled with the GRANT and REVOKE statements
- Implicit authorization is controlled by creating and dropping objects
- Indirect privileges are associated with packages.

Note: A database group name must be 8 characters or less when used in a GRANT or REVOKE statement, or in the Control Center. Even though a database group name longer than 8 characters is accepted, the longer name results in an error message when users belonging to the group access database objects.

Related concepts:
- "Using the system catalog for security issues" on page 161

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Details on controlling access to database objects

The control of access to database objects is through the use of GRANT and REVOKE statements. Implicit access authorization and indirect privileges are also discussed.

Granting privileges

Restrictions:

To grant privileges on most database objects, the user must have SYSADM authority, DBADM authority, or CONTROL privilege on that object; or, the user must hold the privilege WITH GRANT OPTION. Privileges can be granted only on existing objects. To grant CONTROL privilege to someone else, the user must have SYSADM or DBADM authority. To grant DBADM authority, the user must have SYSADM authority.

Procedure:

The GRANT statement allows an authorized user to grant privileges. A privilege can be granted to one or more authorization names in one statement; or to PUBLIC, which makes the privileges available to all users. Note that an authorization name can be either an individual user or a group.

On operating systems where users and groups exist with the same name, you should specify whether you are granting the privilege to the user or group. Both the GRANT and REVOKE statements support the keywords USER and GROUP. If these optional keywords are not used, the database manager checks the operating system security facility to determine whether the authorization name identifies a user or a group. If the authorization name could be both a user and a group, an error is returned.

The following example grants SELECT privileges on the EMPLOYEE table to the user HERON:

```
GRANT SELECT
   ON EMPLOYEE TO USER HERON
```

The following example grants SELECT privileges on the EMPLOYEE table to the group HERON:

```
GRANT SELECT
   ON EMPLOYEE TO GROUP HERON
```

In the Control Center, you can use the Schema Privileges notebook, the Table Space Privileges notebook, and the View Privileges notebook to grant and revoke privileges for these database objects. To open one of these notebooks, follow these steps:

1. In the Control Center, expand the object tree in the left-hand pane until you find the folder containing the objects you want to work with, for example, the Views folder.
2. Click the folder.

   Any existing database objects in this folder are displayed in the (right-hand) contents pane.
3. Right-click the object of interest in the contents pane and select **Privileges** in the pop-up menu.

   The appropriate Privileges notebook opens.

Related concepts:

- “Controlling access to database objects” on page 95

Related tasks:

- “Revoking privileges” on page 97

Related reference:

- “GRANT (Database Authorities) ” on page 1081
- “GRANT (Index Privileges) ” on page 1085
- “GRANT (Package Privileges) ” on page 1087
- “GRANT (Routine Privileges) ” on page 1089
- “GRANT (Schema Privileges) ” on page 1092
- “GRANT (Sequence Privileges) ” on page 1096

- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

Revoking privileges

The REVOKE statement allows authorized users to revoke privileges previously granted to other users.

Restrictions:

To revoke privileges on database objects, you must have DBADM authority, SYSADM authority, or CONTROL privilege on that object. Note that holding a privilege WITH GRANT OPTION is not sufficient to revoke that privilege. To revoke CONTROL privilege from another user, you must have SYSADM or DBADM authority. To revoke DBADM authority, you must have SYSADM authority. Privileges can only be revoked on existing objects.

Note: A user without DBADM authority or CONTROL privilege is not able to revoke a privilege that they granted through their use of the WITH GRANT OPTION. Also, there is no cascade on the revoke to those who have received privileges granted by the person being revoked.

If an explicitly granted table (or view) privilege is revoked from a user with DBADM authority, privileges **will not** be revoked from other views defined on that table. This is because the view privileges are available through the DBADM authority and are not dependent on explicit privileges on the underlying tables.

Procedure:

If a privilege has been granted to both a user and a group with the same name, you must specify the GROUP or USER keyword when revoking the privilege. The following example revokes the SELECT privilege on the EMPLOYEE table from the user HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM USER HERON
```

The following example revokes the SELECT privilege on the EMPLOYEE table from the group HERON:

```
REVOKE SELECT
   ON EMPLOYEE FROM GROUP HERON
```

Note that revoking a privilege from a group may not revoke it from all members of that group. If an individual name has been directly granted a privilege, it will keep it until that privilege is directly revoked.

If a table privilege is revoked from a user, privileges are also revoked on any view created by that user which depends on the revoked table privilege. However, only the privileges implicitly granted by the system are revoked. If a privilege on the view was granted directly by another user, the privilege is still held.

You may have a situation where you want to GRANT a privilege to a group and then REVOKE the privilege from just one member of the group. There are only a couple of ways to do that without receiving the error message SQL0556N:

- You can remove the member from the group; or, create a new group with fewer members and GRANT the privilege to the new group.
- You can REVOKE the privilege from the group and then GRANT it to individual users (authorization IDs).

Note: When CONTROL privilege is revoked from a user on a table or a view, the user continues to have the ability to grant privileges to others. When given CONTROL privilege, the user also receives all other privileges WITH GRANT OPTION. Once CONTROL is revoked, all of the other privileges remain WITH GRANT OPTION until they are explicitly revoked.

All packages that are dependent on revoked privileges are marked invalid, but can be validated if rebound by a user with appropriate authority. Packages can also be rebuilt if the privileges are subsequently granted again to the binder of the application; running the application will trigger a successful implicit rebind. If privileges are revoked from PUBLIC, all packages bound by users having only been able to bind based on PUBLIC privileges are invalidated. If DBADM authority is revoked from a user, all packages bound by that user are invalidated including those associated with database utilities. Attempting to use a package that has been marked invalid causes the system to attempt to rebind the package. If this rebind attempt fails, an error occurs (SQLCODE -727). In this case, the packages must be explicitly rebound by a user with:
- Authority to rebind the packages
- Appropriate authority for the objects used within the packages

These packages should be rebound at the time the privileges are revoked.

If you define a trigger or SQL function based on one or more privileges and you lose one or more of these privileges, the trigger or SQL function cannot be used.

Related tasks:
- “Granting privileges” on page 95

Related reference:
- “REVOKE (Database Authorities) ” on page 1120
- “REVOKE (Index Privileges) ” on page 1125
- “REVOKE (Package Privileges) ” on page 1126
- “REVOKE (Routine Privileges) ” on page 1129
- “REVOKE (Schema Privileges) ” on page 1132
- “REVOKE (Server Privileges) ” on page 1136
- “REVOKE (Table Space Privileges) ” on page 1139
- “REVOKE (Table, View, or Nickname Privileges) ” on page 1141

Managing implicit authorizations by creating and dropping objects

Procedure:

The database manager implicitly grants certain privileges to a user creates a database object such as a table or a package. Privileges are also granted when objects are created by users with SYSADM or DBADM authority. Similarly, privileges are removed when an object is dropped.

When the created object is a table, nickname, index, or package, the user receives CONTROL privilege on the object. When the object is a view, the CONTROL privilege for the view is granted implicitly only if the user has CONTROL privilege for all tables, views, and nicknames referenced in the view definition.

When the object explicitly created is a schema, the schema owner is given ALTERIN, CREATEIN, and DROPIN privileges WITH GRANT OPTION. An implicitly created schema has CREATEIN granted to PUBLIC.

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Establishing ownership of a package

Procedure:

The BIND and PRECOMPILE commands create or change an application package. On either one, use the OWNER option to name the owner of the resulting package. There are simple rules for naming the owner of a package:
- Any user can name themselves as the owner. This is the default if the `OWNER` option is not specified.
- An ID with SYSADM or DBADM authority can name any authorization ID as the owner using the OWNER option.

Not all operating systems that can bind a package using DB2 database products support the OWNER option.

Related reference:
- "BIND" on page 441
- "PRECOMPILE " on page 635

Indirect privileges through a package

Access to data within a database can be requested by application programs, as well as by persons engaged in an interactive workstation session. A package contains statements that allow users to perform a variety of actions on many database objects. Each of these actions requires one or more privileges.

Privileges granted to individuals binding the package and to PUBLIC are used for authorization checking when static SQL and XQuery statements are bound. Privileges granted through groups are *not* used for authorization checking when static SQL and XQuery statements are bound. The user with a valid *authID* who binds a package must either have been explicitly granted all the privileges required to execute the static SQL or XQuery statements in the package or have been implicitly granted the necessary privileges through PUBLIC unless VALIDATE RUN was specified when binding the package. If VALIDATE RUN was specified at BIND time, all authorization failures for any static SQL or XQuery statements within this package will not cause the BIND to fail, and those SQL or XQuery statements are revalidated at run time. PUBLIC, group, and user privileges *are all* used when checking to ensure the user has the appropriate authorization (BIND or BINDADD privilege) to bind the package.

Packages may include both static and dynamic SQL and XQuery statements. To process a package with static queries, a user need only have EXECUTE privilege on the package. This user can then indirectly obtain the privileges of the package binder for any static queries in the package but only within the restrictions imposed by the package.

If the package includes dynamic SQL or XQuery statements, the required privileges depend on the value that was specified for DYNAMICRULES when the package was precompiled or bound. For more information, see the topic that describes the effect of DYNAMICRULES on dynamic queries.

Related concepts:
- "Indirect privileges through a package containing nicknames" on page 100
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 1308

Related reference:
- "BIND" on page 441

Indirect privileges through a package containing nicknames

When a package contains references to nicknames, authorization processing for package creators and package users is slightly more complex. When a package creator successfully binds packages that contain nicknames, the package creator does not have to pass authentication checking or privilege checking for the tables and views that the nicknames reference at the data source. However, the package executor must pass authentication and authorization checking at data sources.

For example, assume that a package creator's .SQC file contains several SQL or XQuery statements. One static statement references a local table. Another dynamic statement references a nickname. When the package is bound, the package creator's authid is used to verify privileges for the local table and the nickname, but no checking is done for the data source objects that the nickname identifies. When another user executes the package, assuming they have the EXECUTE privilege for that package, that user does not have to pass any additional privilege checking for the statement referencing the table. However, for the statement referencing the nickname, the user executing the package must pass authentication checking and privilege checking at the data source.

When the .SQC file contains only dynamic SQL and XQuery statements and a mixture of table and nickname references, DB2 database authorization checking for local objects and nicknames is similar. Package users must pass privilege checking for any local objects (tables, views) within the statements and also pass privilege checking for nickname objects (package users must pass authentication and privilege checking at the data source containing the objects that the nicknames identify). In both cases, users of the package must have the EXECUTE privilege.

The ID and password of the package executor is used for all data source authentication and privilege processing. This information can be changed by creating a user mapping.

Note: Nicknames cannot be specified in static SQL and XQuery statements. Do not use the DYNAMICRULES option (set to BIND) with packages containing nicknames.

It is possible that packages containing nicknames might require additional authorization steps because DB2 database uses dynamic SQL when communicating with DB2 Family data sources. The authorization ID running the package at the data source must have the appropriate authority to execute the package dynamically at that data source.

Related concepts:
- "Indirect privileges through a package" on page 99

Controlling access to data with views

A view provides a means of controlling access or extending privileges to a table by allowing:
- Access only to designated columns of the table.

  For users and application programs that require access only to specific columns of a table, an authorized user can create a view to limit the columns addressed only to those required.
- Access only to a subset of the rows of the table.

  By specifying a WHERE clause in the subquery of a view definition, an authorized user can limit the rows addressed through a view.
- Access only to a subset of the rows or columns in data source tables or views. If you are accessing data sources through nicknames, you can create local DB2 database views that reference nicknames. These views can reference nicknames from one or many data sources.

  Note: Because you can create a view that contains nickname references for more than one data source, your users can access data in multiple data sources from one view. These views are called *multi-location views*. Such views are useful when joining information in columns of sensitive tables across a distributed environment or when individual users lack the privileges needed at data sources for specific objects.

To create a view, a user must have SYSADM authority, DBADM authority, or CONTROL or SELECT privilege for each table, view, or nickname referenced in the view definition. The user must also be able to create an object in the schema specified for the view. That is, CREATEIN privilege for an existing schema or IMPLICIT_SCHEMA authority on the database if the schema does not already exist.

If you are creating views that reference nicknames, you do not need additional authority on the data source objects (tables and views) referenced by nicknames in the view; however, users of the view must have SELECT authority or the equivalent authorization level for the underlying data source objects when they access the view.

If your users do not have the proper authority at the data source for underlying objects (tables and views), you can:
1. Create a data source view over those columns in the data source table that are OK for the user to access
2. Grant the SELECT privilege on this view to users
3. Create a nickname to reference the view

Users can then access the columns by issuing a SELECT statement that references the new nickname.

The following scenario provides a more detailed example of how views can be used to restrict access to information.

Many people might require access to information in the STAFF table, for different reasons. For example:

- The personnel department needs to be able to update and look at the entire table.

  This requirement can be easily met by granting SELECT and UPDATE privileges on the STAFF table to the group PERSONNL:

  ```
  GRANT SELECT,UPDATE ON TABLE STAFF TO GROUP PERSONNL
  ```
- Individual department managers need to look at the salary information for their employees.

  This requirement can be met by creating a view for each department manager. For example, the following view can be created for the manager of department number 51:

  ```
  CREATE VIEW EMP051 AS
     SELECT NAME,SALARY,JOB FROM STAFF
     WHERE DEPT=51
  GRANT SELECT ON TABLE EMP051 TO JANE
  ```

  The manager with the authorization name JANE would query the EMP051 view just like the STAFF table. When accessing the EMP051 view of the STAFF table, this manager views the following information:

NAME	SALARY	JOB
Fraye	45150.0	Mgr
Williams	37156.5	Sales
Smith	35654.5	Sales
Lundquist	26369.8	Clerk
Wheeler	22460.0	Clerk

- All users need to be able to locate other employees. This requirement can be met by creating a view on the NAME column of the STAFF table and the LOCATION column of the ORG table, and by joining the two tables on their respective DEPT and DEPTNUMB columns:

  ```
  CREATE VIEW EMPLOCS AS
     SELECT NAME, LOCATION FROM STAFF, ORG
     WHERE STAFF.DEPT=ORG.DEPTNUMB
  GRANT SELECT ON TABLE EMPLOCS TO PUBLIC
  ```

  Users who access the employee location view will see the following information:

NAME	LOCATION
Molinare	New York
Lu	New York
Daniels	New York
Jones	New York
Hanes	Boston
Rothman	Boston
Ngan	Boston
Kermisch	Boston
Sanders	Washington

NAME	LOCATION
Pernal	Washington
James	Washington
Sneider	Washington
Marenghi	Atlanta
O’Brien	Atlanta
Quigley	Atlanta
Naughton	Atlanta
Abrahams	Atlanta
Koonitz	Chicago
Plotz	Chicago
Yamaguchi	Chicago
Scoutten	Chicago
Fraye	Dallas
Williams	Dallas
Smith	Dallas
Lundquist	Dallas
Wheeler	Dallas
Lea	San Francisco
Wilson	San Francisco
Graham	San Francisco
Gonzales	San Francisco
Burke	San Francisco
Quill	Denver
Davis	Denver
Edwards	Denver
Gafney	Denver

Related tasks:
- “Creating a view” on page 311
- “Granting privileges” on page 95

Monitoring access to data using the audit facility

The DB2 database audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. While not a facility that prevents access to data, the audit facility can monitor and keep a record of attempts to access or modify data objects.

SYSADM authority is required to use the audit facility administrator tool, `db2audit`.

Related concepts:
- “Introduction to the DB2 database audit facility” on page 197

Data encryption

One part of your security plan may involve encrypting your data. To do this, you can use encryption and decryption built-in functions: ENCRYPT, DECRYPT_BIN, DECRYPT_CHAR, and GETHINT.

The ENCRYPT function encrypts data using a password-based encryption method. These functions also allow you to encapsulate a password hint. The password hint is embedded in the encrypted data. Once encrypted, the only way to decrypt the data is by using the correct password. Developers that choose to use these functions should plan for the management of forgotten passwords and unusable data.

The result of the ENCRYPT functions is VARCHAR FOR BIT DATA (with a limit of 32 631).

Only CHAR, VARCHAR, and FOR BIT DATA can be encrypted.

The DECRYPT_BIN and DECRYPT_CHAR functions decrypt data using password-based decryption.

DECRYPT_BIN always returns VARCHAR FOR BIT DATA while DECRYPT_CHAR always returns VARCHAR. Since the first argument may be CHAR FOR BIT DATA or VARCHAR FOR BIT DATA, there are cases where the result is not the same as the first argument.

The length of the result depends on the bytes to the next 8 byte boundary. The length of the result could be the length of the data argument plus 40 plus the number of bytes to the next 8 byte boundary when the optional hint parameter is specified. Or, the length of the result could be the length of the data argument plus 8 plus the number of bytes to the next 8 byte boundary when the optional hint parameter is not specified.

The GETHINT function returns an encapsulated password hint. A password hint is a phrase that will help data owners remember passwords. For example, the word “Ocean” can be used as a hint to remember the password "Pacific".

The password that is used to encrypt the data is determined in one of two ways:

- Password Argument. The password is a string that is explicitly passed when the ENCRYPT function is invoked. The data is encrypted and decrypted with the given password.
- Encryption password special register. The SET ENCRYPTION PASSWORD statement encrypts the password value and sends the encrypted password to the database manager to store in a special register. ENCRYPT, DECRYPT_BIN and DECRYPT_CHAR functions invoked without a password parameter use the value in the ENCRYPTION PASSWORD special register. The ENCRYPTION PASSWORD special register is only stored in encrypted form.

  The initial or default value for the special register is an empty string.

Valid lengths for passwords are between 6 and 127 inclusive. Valid lengths for hints are between 0 and 32 inclusive.

Related reference:

- “DECRYPT_BIN and DECRYPT_CHAR ” on page 1265

- "ENCRYPT " on page 1266
- "GETHINT " on page 1268
- "SET ENCRYPTION PASSWORD statement" in *SQL Reference, Volume 2*

Tasks and required authorizations

Not all organizations divide job responsibilities in the same manner. Table 8 lists some other common job titles, the tasks that usually accompany them, and the authorities or privileges that are needed to carry out those tasks.

Table 8. Common Job Titles, Tasks, and Required Authorization

JOB TITLE	TASKS	REQUIRED AUTHORIZATION
Department Administrator	Oversees the departmental system; creates databases	SYSCTRL authority. SYSADM authority if the department has its own instance.
Security Administrator	Authorizes other users for some or all authorizations and privileges	SYSADM or DBADM authority.
Database Administrator	Designs, develops, operates, safeguards, and maintains one or more databases	DBADM and SYSMAINT authority over one or more databases. SYSCTRL authority in some cases.
System Operator	Monitors the database and carries out backup functions	SYSMAINT authority.
Application Programmer	Develops and tests the database manager application programs; may also create tables of test data	BINDADD, BIND on an existing package, CONNECT and CREATETAB on one or more databases, some specific schema privileges, and a list of privileges on some tables. CREATE_EXTERNAL_ROUTINE may also be required.
User Analyst	Defines the data requirements for an application program by examining the system catalog views	SELECT on the catalog views; CONNECT on one or more databases.
Program End User	Executes an application program	EXECUTE on the package; CONNECT on one or more databases. See the note following this table.
Information Center Consultant	Defines the data requirements for a query user; provides the data by creating tables and views and by granting access to database objects	DBADM authority over one or more databases.
Query User	Issues SQL statements to retrieve, add, delete, or change data; may save results as tables	CONNECT on one or more databases; CREATEIN on the schema of the tables and views being created; and, SELECT, INSERT, UPDATE, DELETE on some tables and views.

Note: If an application program contains dynamic SQL statements, the Program End User may need other privileges in addition to EXECUTE and CONNECT (such as SELECT, INSERT, DELETE, and UPDATE).

Related concepts:

- “Database administration authority (DBADM)” on page 76
- “Database authorities” on page 74
- “LOAD authority” on page 77
- “System administration authority (SYSADM)” on page 71
- “System control authority (SYSCTRL)” on page 72
- “System maintenance authority (SYSMAINT)” on page 72

Related tasks:
- “Granting privileges” on page 95
- “Revoking privileges” on page 97

Acquiring Windows users’ group information using an access token

An access token is an object that describes the security context of a process or thread. The information in an access token includes the identity and privileges of the user account associated with the process or thread.

When you log on, the system verifies your password by comparing it with information stored in a security database. If the password is authenticated, the system produces an access token. Every process run on your behalf uses a copy of this access token.

An access token can also be acquired based on cached credentials. Once you have been authenticated to the system, your credentials are cached by the operating system. The access token of the last logon can be referenced in the cache when it is not possible to contact the domain controller.

The access token includes information about all of the groups you belong to: local groups and various domain groups (global groups, domain local groups, and universal groups).

Note: Group lookup using client authentication is not supported using a remote connection even though access token support is enabled.

To enable access token support, you must use the `db2set` command to update the DB2_GRP_LOOKUP registry variable. Your choices when updating this registry variable include:

- TOKEN

  This choice enables access token support to lookup all groups that the user belongs to at the location where the user account is defined. This location is typically either at the domain or local to the DB2 database server.
- TOKENLOCAL

  This choice enables access token support to lookup all local groups that the user belongs to on the DB2 database server.
- TOKENDOMAIN

  This choice enables access token support to lookup all domain groups that the user belongs to on the domain.

When enabling access token support, there are several limitations that affect your account management infrastructure. When this support is enabled, the DB2 database system collects group information about the user who is connecting to the database. Subsequent operations after a successful CONNECT or ATTACH request that have dependencies on other authorization IDs will still need to use conventional group enumeration. The access token advantages of nested global groups, domain local groups, and cached credentials will not be available. For example, if, after a connection, the SET SESSION_USER is used to run under another authorization ID, only the conventional group enumeration is used to check what rights are given to the new authorization ID for the session. You will still need to grant and revoke explicit privileges to individual authorization IDs known to the DB2 database system, as opposed to the granting and revoking of privileges to groups to which the authorization IDs belongs.

If you intend to assign groups to SYSADM, SYSMAINT, or SYSCTRL, you need to ensure that the assigned groups are not nested global groups, nor domain local groups, and then the cached credential capability is not needed.

You should consider using the DB2_GRP_LOOKUP registry variable and specify the group lookup location to indicate where the DB2 database system should look up groups using the conventional group enumeration methodology. For example,

```
db2set DB2_GRP_LOOKUP=LOCAL,TOKENLOCAL
```

This enables the access token support for enumerating local groups. Group lookup for an authorization ID different from the connected user is performed at the DB2 database server.

```
db2set DB2_GRP_LOOKUP=,TOKEN
```

This enables the access token support for enumerating groups at the location where the user ID is defined. Group lookup for an authorization ID different from the connected user is performed where the user ID is defined.

```
db2set DB2_GRP_LOOKUP=DOMAIN,TOKENDOMAIN
```

This enables the access token support for enumerating domain groups. Group lookup for an authorization ID different from the connected user is performed where the user ID is defined.

Applications using dynamic queries in a package bound using DYNAMICRULES RUN (which is the default) is run under the privileges of the person who runs the application. In this case, the already mentioned limitations do not apply. This would include applications written to use JDBC and DB2 CLI.

Access token support can be enabled with all authentications types except CLIENT authentication.

Related concepts:

- "Security issues when installing the DB2 database manager" on page 87

Details on security based on operating system

Each operating system provides ways to manage security. Some of the security issues associated with the operating systems are discussed in this section.

Windows platform security considerations for users

System Administration (SYSADM) authority is granted to any valid DB2 database user account which belongs to the local Administrators group on the machine where the account is defined.

By default in a Windows domain environment, only domain users that belong to the Administrators group at the Domain Controller have SYSADM authority on an instance. Since DB2 always performs authorization at the machine where the account is defined, adding a domain user to the local Administrators group on the server does not grant the domain user SYSADM authority to the group.

Note: In a domain environment such as is found in Windows, DB2 only authenticates the first 64 groups that meet the requirements and restrictions, and to which a user ID belongs. You may have more than 64 groups.

To avoid adding a domain user to the Administrators group at the PDC, you should create a global group and add the users (both domain and local) that you want to grant SYSADM authority. To do this, enter the following commands:

```
DB2STOP
DB2 UPDATE DBM CFG USING SYSADM_GROUP global_group
DB2START
```

Related concepts:

- "UNIX platform security considerations for users" on page 108

UNIX platform security considerations for users

The DB2 database does not support root acting directly as a database administrator. You should use `su - <instance owner>` as the database administrator.

For security reasons, we recommend you do not use the instance name as the Fenced ID. However, if you are not planning to use fenced UDFs or stored procedures, you can set the Fenced ID to the instance name instead of creating another user ID.

The recommendation is to create a user ID that will be recognized as being associated with this group. The user for fenced UDFs and stored procedures is specified as a parameter of the instance creation script (`db2icrt ... -u <FencedID>`). This is not required if you install the DB2 Clients or the DB2 Software Developer's Kit.

Related concepts:

- "Windows platform security considerations for users" on page 107

Chapter 14. Label-based access control (LBAC)

This section explains what you need to know in order to use label-based access control (LBAC).

Label-based access control (LBAC) overview

What LBAC does:

Label-based access control (LBAC) greatly increases the control you have over who can access your data. LBAC lets you decide exactly who has write access and who has read access to individual rows and individual columns.

The LBAC capability is very configurable and can be tailored to match your particular security environment. All LBAC configuration is performed by a *security administrator*, which is a user that has been granted the SECADM authority by the system administrator.

A security administrator configures the LBAC system by creating security policies. A *security policy* describes the criteria that will be used to decide who has access to what data. Only one security policy can be used to protect any one table but different tables can be protected by different security policies.

After creating a security policy, a security administrator creates objects, called *security labels* that are part of that policy. Exactly what makes up a security label is determined by the security policy and can be configured to represent the criteria that your organization uses to decide who should have access to particular data items. If you decide, for instance, that you want to look at a person's position in the company and what projects they are part of to decide what data they should see, then you can configure your security labels so that each label can include that information. LBAC is flexible enough to let you set up anything from very complicated criteria, to a very simple system where each label represents either a "high" or a "low" level of trust.

Once created, a security label can be associated with individual columns and rows in a table to protect the data held there. Data that is protected by a security label is called *protected data*. A security administrator allows users access to protected data by granting them security labels. When a user tries to access protected data, that user's security label is compared to the security label protecting the data. The protecting label will block some security labels and not block others.

A user is allowed to hold security labels for multiple security policies at once. For any given security policy, however, a user can hold at most one label for read access and one label for write access.

A security administrator can also grant exemptions to users. An *exemption* allows you to access protected data that your security labels might otherwise prevent you from accessing. Together your security labels and exemptions are called your *LBAC credentials*.

If you try to access a protected column that your LBAC credentials do not allow you to access then the access will fail and you will get an error message.

If you try to read protected rows that your LBAC credentials do not allow you to read then DB2 acts as if those rows do not exist. Those rows cannot be selected as part of any SQL statement that you run, including SELECT, UPDATE, or DELETE. Even the aggregate functions ignore rows that your LBAC credentials do not allow you to read. The COUNT(*) function, for example, will return a count only of the rows that you have read access to.

Views and LBAC:

You can define a view on a protected table the same way you can define one on a non-protected table. When such a view is accessed the LBAC protection on the underlying table is enforced. The LBAC credentials used are those of the session authorization ID. Two users accessing the same view might see different rows depending on their LBAC credentials.

Referential integrity constraints and LBAC:

The following rules explain how LBAC rules are enforced in the presence of referential integrity constraints:

- **Rule 1**: The LBAC read access rules are NOT applied for internally generated scans of child tables. This is to avoid having orphan children.
- **Rule 2**: The LBAC read access rules are NOT applied for internally generated scans of parent tables
- **Rule 3**: The LBAC write rules are applied when a CASCADE operation is performed on child tables. For example, If a user deletes a parent, but cannot delete any of the children because of an LBAC write rule violation, then the delete should be rolled-back and an error raised.

What LBAC does not do:

- LBAC will never allow access to data that is forbidden by discretionary access control.

  Example: If you do not have permission to read from a table then you will not be allowed to read data from that table--even the rows and columns to which LBAC would otherwise allow you access.
- Your LBAC credentials only limit your access to protected data. They have no effect on your access to unprotected data.
- LBAC credentials are not checked when you drop a table or a database, even if the table or database contains protected data.
- LBAC credentials are not checked when you back up your data. If you can run a backup on a table, which rows are backed up is not limited in any way by the LBAC protection on the data. Also, data on the backup media is not protected by LBAC. Only data in the database is protected.
- LBAC cannot be used to protect any of the following types of tables:
  - A materialized query table (MQT)
  - A table that a materialized query table (MQT) depends on
  - A staging table
  - A table that a staging table depends on
  - A typed table
- LBAC protection cannot be applied to a nickname.

LBAC tutorial:

A tutorial leading you through the basics of using LBAC is available online. The tutorial is part of the IBM developerWorks website (http://www.ibm.com/developerworks/db2) and is called DB2 Label-Based Access Control, a practical guide.

Related concepts:
- "Database authorities" on page 74
- "LBAC security label components overview" on page 112
- "LBAC security labels" on page 118
- "LBAC security policies" on page 111

LBAC security policies

A *security policy* is a database object that is part of label-based access control (LBAC). A security policy includes this information:
- What security label components will be used in the security labels that are part of the policy
- What rules will be used when comparing those security label components
- Which of certain optional behaviors will be used when accessing data protected by the policy

Every protected table must have one and only one security policy associated with it. Rows and columns in that table can only be protected with security labels that are part of that security policy and all access of protected data follows the rules of that policy. You can have multiple security policies in a single database but you cannot have more than one security policy protecting any given table.

Creating a security policy:

You must be a security administrator to create a security policy. You create a security policy with the SQL statement CREATE SECURITY POLICY. The security label components listed in a security policy must be created before the CREATE SECURITY POLICY statement is executed. The order in which the components are listed when a security policy is created does not indicate any sort of precedence or other relationship among the components but it is important to know the order when creating security labels with built-in functions like SECLABEL.

Altering a security policy:

Security policies cannot be altered. The only way to change a security policy is to drop it and re-create it.

Dropping a security policy:

You must be a security administrator to drop a security policy. You drop a security policy using the SQL statement DROP.

You cannot drop a security policy if it is associated with (added to) any table.

Related concepts:
- "Label-based access control (LBAC) overview" on page 109

Related reference:

- “CREATE SECURITY LABEL COMPONENT ” on page 952
- “CREATE SECURITY POLICY ” on page 955
- “DROP ” on page 1054

LBAC security label components

This section explains what you need to know in order to use LBAC security label components.

LBAC security label components overview

A *security label component* is a database object that is part of label-based access control (LBAC). You use security label components to model your organization’s security structure.

A component can represent any criteria that you might use to decide if a user should have access to a given piece of data. Typical examples of such criteria include:

- How well trusted the user is
- What department the user is in
- Whether the user is involved in a particular project

Example: If you want the department that a user is in to affect which data they can access, you could create a component named dept and define elements for that component that name the various departments in your company. You would then include the component dept in your security policy.

An *element* of a security label component is one particular "setting" that is allowed for that component.

Example: A security label component that represents a level of trust might have the four elements: Top Secret, Secret, Classified, and Unclassified.

Creating a security label component:

You must be a security administrator to create a security label component. You create security label components with the SQL statement CREATE SECURITY LABEL COMPONENT.

When you create a security label component you must provide:

- A name for the component
- What type of component it is (ARRAY, TREE, or SET)
- A complete list of allowed elements
- For types ARRAY and TREE you must describe how each element fits into the structure of the component

Types of components:

There are three types of security label components:

- TREE: Each element represents a node in a tree structure
- ARRAY: Each element represents a point on a linear scale
- SET: Each element represents one member of a set

The types are used to model the different ways in which elements can relate to each other. For example, if you are creating a component to describe one or more departments in a company you would probably want to use a component type of TREE because most business structures are in the form of a tree. If you are creating a component to represent the level of trust that a person has, you would probably use a component of type ARRAY because for any two levels of trust, one will always be higher than the other.

The details of each type, including detailed descriptions of the relationships that the elements can have with each other, are described in their own section.

Altering security label components:

Security label components cannot be altered. The only way to change a security label component is to drop it and re-create it.

Dropping a security label component:

You must be a security administrator to drop a security label component. You drop a security label component with the SQL statement DROP.

Related concepts:
- "LBAC security label component type: ARRAY" on page 113
- "LBAC security label component type: SET" on page 113
- "LBAC security label component type: TREE" on page 114

Related reference:
- "CREATE SECURITY LABEL COMPONENT " on page 952
- "DROP " on page 1054

LBAC security label component type: SET

SET is one type of security label component that can be used in a label-based access control (LBAC) security policy. Components of this type are unordered lists of elements. The only comparison that can be made for elements of this type of component is whether or not a given element is in the list.

Related concepts:
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security label components overview" on page 112
- "LBAC security policies" on page 111

Related reference:
- "CREATE SECURITY LABEL COMPONENT " on page 952

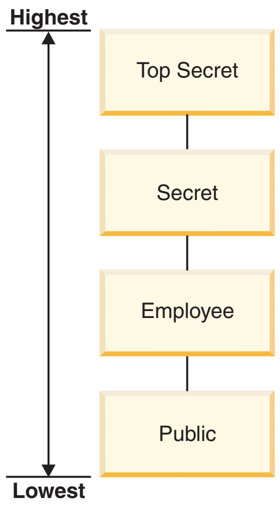
LBAC security label component type: ARRAY

ARRAY is one type of security label component. In this type of component the order in which the elements are listed when the component is created defines a scale with the first element listed being the highest value and the last being the lowest.

Example: If the component mycomp is defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
  ARRAY [ 'Top Secret', 'Secret', 'Employee', 'Public' ]
```

Then the elements are treated as if they are organized in a structure like this:



In a component of type ARRAY, the elements can have these sorts of relationships to each other:

Higher than
Element A is higher than element B if element A is listed earlier in the ARRAY clause than element B.

Lower than
Element A is lower than element B if element A is listed later in the ARRAY clause than element B

Related concepts:
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security label components overview" on page 112
- "LBAC security policies" on page 111

Related reference:
- "CREATE SECURITY LABEL COMPONENT " on page 952

LBAC security label component type: TREE

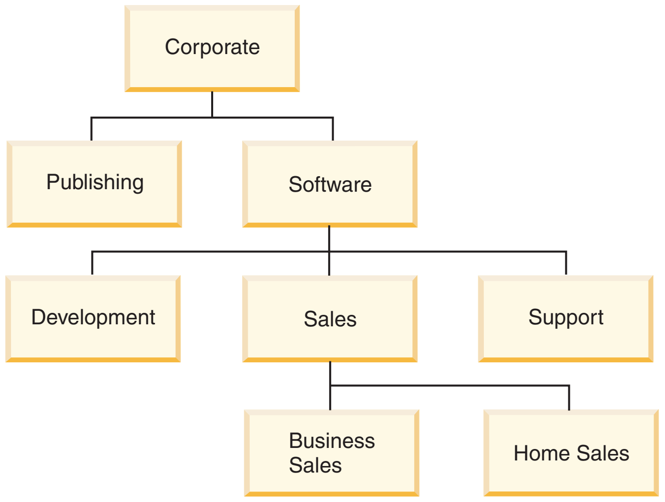
Components of type TREE:

TREE is one type of security label component that can be used in a label-based access control (LBAC) security policy. In this type of component the elements are treated as if they are arranged in a tree structure. When you specify an element that is part of a component of type TREE you must also specify which other element it is under. The one exception is the first element which must be specified as being the ROOT of the tree. This allows you to organize the elements in a tree structure.

Example: If the component mycomp is defined this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
TREE (
    'Corporate'      ROOT,
    'Publishing'     UNDER 'Corporate',
    'Software'       UNDER 'Corporate',
    'Development'    UNDER 'Software',
    'Sales'          UNDER 'Software',
    'Support'        UNDER 'Software'
    'Business Sales' UNDER 'Sales'
    'Home Sales'     UNDER 'Sales'
)
```

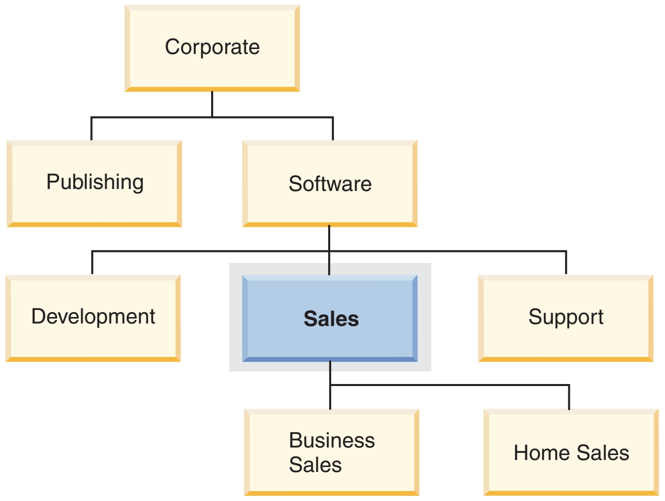
Then the elements are treated as if they are organized in a tree structure like this:



In a component of type TREE, the elements can have these types of relationships to each other:

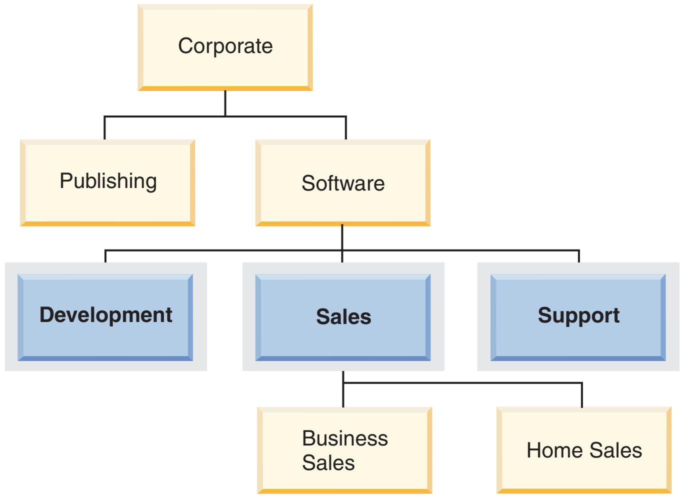
Parent Element A is a parent of element B if element B is UNDER element A.

Example: This diagram shows the parent of the Business Sales element:



Child Element A is a child of element B if element A is UNDER element B.

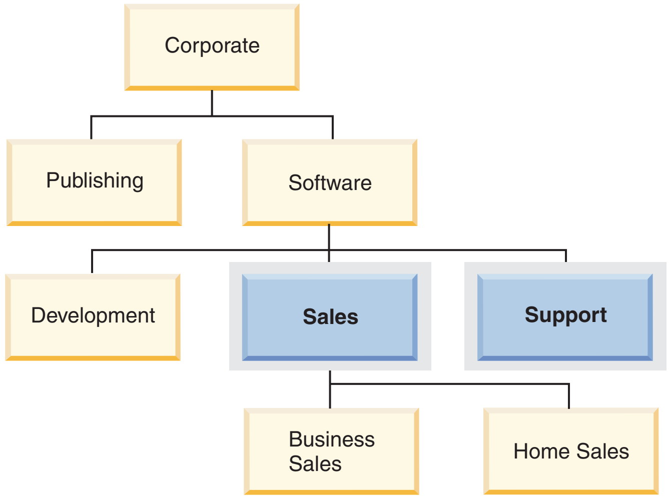
Example: This diagram shows the children of the Software element:



Sibling

Two elements are siblings of each other if they have the same parent.

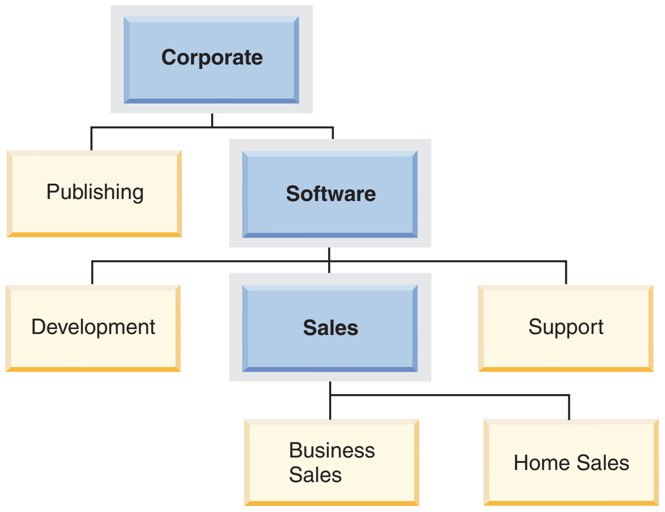
Example: This diagram shows the siblings of the Development element:



Ancestor

Element A is an ancestor of element B if it is the parent of B, or if it is the parent of the parent of B, and so on. The root element is an ancestor of all other elements in the tree.

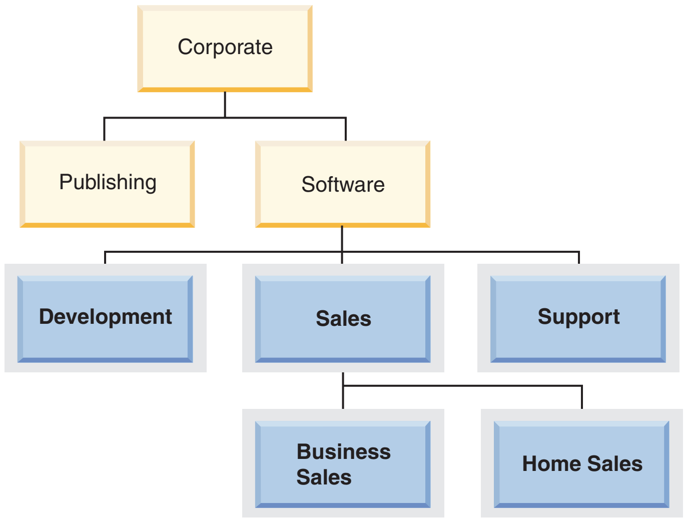
Example: This diagram shows the ancestors of the Home Sales element:



Descendent

Element A is a descendent of element B if it is the child of B, or if it is the child of a child of B, and so on.

Example: This diagram shows the descendents of the Software element:



Related concepts:

- "Label-based access control (LBAC) overview" on page 109
- "LBAC security label components overview" on page 112
- "LBAC security policies" on page 111

Related reference:

- "CREATE SECURITY LABEL COMPONENT " on page 952

LBAC security labels

In label-based access control (LBAC) a *security label* is a database object that describes a certain set of security criteria. Security labels are applied to data in order to protect the data. They are granted to users to allow them to access protected data. When a user tries to access protected data, their security label is compared to the security label that is protecting the data. The protecting security label will block some security labels and not block others. If a user's security label is blocked then the user cannot access the data.

Every security label is part of exactly one security policy and includes one value for each component in that security policy. A *value* in the context of a security label component is a list of zero or more of the elements allowed by that component. Values for ARRAY type components can contain zero or one element, values for other types can have zero or more elements. A value that does not include any elements is called an *empty value*.

Example: If a TREE type component has the three elements Human Resources, Sales, and Shipping then these are some of the valid values for that component:

- Human Resources (or any of the elements by itself)
- Human Resources, Shipping (or any other combination of the elements as long as no element is included more than once)
- *An empty value*

Whether a particular security label will block another is determined by the values of each component in the labels and the LBAC rule set that is specified in the security policy of the table. The details of how the comparison is made are given in the section How LBAC security labels are compared.

When security labels are converted to a text string they use the format described in the section Format for security label values.

Creating security labels:

You must be a security administrator to create a security label. You create a security label with the SQL statement CREATE SECURITY LABEL. When you create a security label you provide:

- A name for the label
- The security policy that the label is part of
- Values for one or more of the components included in the security policy

Any components for which a value is not specified is assumed to have an empty value. A security label must have at least one non-empty value.

Altering security labels:

Security labels cannot be altered. The only way to change a security label is to drop it and re-create it.

Dropping security labels:

You must be a security administrator to drop a security label. You drop a security label with the SQL statement DROP. You cannot drop a security label that is being used to protect data anywhere in the database or that is currently held by one or more users.

Granting security labels:

You must be a security administrator to grant a security label to a user. You grant a security label to a user with the SQL statement GRANT SECURITY LABEL. When you grant a security label you can grant it for read access, for write access, or for both read and write access. A user cannot hold two security labels from the same security policy for the same type of access.

Revoking security labels:

You must be a security administrator to revoke a security label from a user. To revoke a security label, use the SQL statement REVOKE SECURITY LABEL.

Data types compatible with security labels:

Security labels have a data type of SYSPROC.DB2SECURITYLABEL. Data conversion is supported between SYSPROC.DB2SECURITYLABEL and VARCHAR(128) FOR BIT DATA.

Related concepts:
- "How LBAC security labels are compared" on page 120
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security label components overview" on page 112
- "Protection of data using LBAC" on page 128

Related reference:
- "CREATE SECURITY LABEL " on page 951
- "Format for security label values" on page 119

Format for security label values

Sometimes the values in a security label are represented in the form of a character string, for example when using the built-in function SECLABEL. When representing the values in a security label as a string this format is used.
- The values of the components are listed from left to right in the same order that the components are listed in the CREATE SECURITY POLICY statement for the security policy
- An element is represented by the name of that element
- Elements for different components are separated by a colon (:)
- If more than one element are given for the same component the elements are enclosed in parentheses (()) and are separated by a comma (,)
- Empty values are represented by a set of empty parentheses (())

Example: A security label is part of a security policy that has these three components in this order: Level, Department, and Projects. The security label has these values:

Table 9.

Component	Values
Level	Secret
Department	*Empty value*
Projects	• Epsilon 37 • Megaphone • Cloverleaf

This security label values look like this as a string:

```
'Secret:():(Epsilon 37,Megaphone,Cloverleaf)'
```

Related concepts:
- "How LBAC security labels are compared" on page 120
- "LBAC security label components overview" on page 112
- "LBAC security labels" on page 118

How LBAC security labels are compared

When you try to access data protected by label-based access control (LBAC), Your LBAC credentials are compared to one or more security labels to see if the access is blocked. Your LBAC credentials are any security labels you hold plus any exemptions that you hold.

There are only two types of comparison that can be made. Your LBAC credentials can be compared to a single security label for read access or your LBAC credentials compared to a single security label for write access. Updating and deleting are treated as being a read followed by a write. When an operation requires multiple comparisons to be made, each is made separately.

Which of your security labels is used:

Even though you might hold multiple security labels only one is compared to the protecting security label. The label used is the one that meets these criteria:
- It is part of the security policy that is protecting the table being accessed.
- It was granted for the type of access (read or write).

If you do not have a security label that meets these criteria then a default security label is assumed that has empty values for all components.

How the comparison is made:

Security labels are compared component by component. If a security label does not have a value for one of the components then an empty value is assumed. As each component is examined, the appropriate rules of the LBAC rule set are used to decide if the elements in your value for that component should be blocked by the elements in the value for the same component in the protecting label. If any of your values are blocked then your LBAC credentials are blocked by the protecting security label.

The LBAC rule set used in the comparison is designated in the security policy. To find out what the rules are and when each one is used, see the description of that rule set.

How exemptions affect comparisons:

If you hold an exemption for the rule that is being used to compare two values then that comparison is not done and the protecting value is assumed not to block the value in your security label.

Example: The LBAC rule set is DB2LBACRULES and the security policy has two components. One component is of type ARRAY and the other is of type TREE. The user has been granted an exemption on the rule DB2LBACREADTREE, which is the rule used for read access when comparing values of components of type TREE. If the user attempts to read protected data then whatever value the user has for the TREE component, even if it is an empty value, will not block access because that rule is not used. Whether the user can read the data depends entirely on the values of the ARRAY component of the labels.

Related concepts:
- "Label-based access control (LBAC) overview" on page 109
- "LBAC rule exemptions" on page 126
- "LBAC rule set: DB2LBACRULES" on page 122
- "LBAC rule sets overview" on page 121
- "LBAC security labels" on page 118
- "LBAC security policies" on page 111

LBAC rule sets

This section explains LBAC rule sets in general and also the details of the DB2LBACRULES rule set.

LBAC rule sets overview

An LBAC rule set is a predefined set of rules that are used when comparing security labels. When the values of a two security labels are being compared, one or more of the rules in the rule set will be used to determine if one value blocks another.

Each LBAC rule set is identified by a unique name. When you create a security policy you must specify the LBAC rule set that will be used with that policy. Any comparison of security labels that are part of that policy will use that LBAC rule set.

Each rule in a rule set is also identified by a unique name. You use the name of a rule when you are granting an exemption on that rule.

How many rules are in a set and when each rule is used can vary from rule set to rule set.

There is currently only one supported LBAC rule set. The name of that rule set is DB2LBACRULES.

Related concepts:
- "Label-based access control (LBAC) overview" on page 109
- "LBAC rule set: DB2LBACRULES" on page 122

LBAC rule set: DB2LBACRULES

This LBAC rule set provides a traditional set of rules for comparing the values of security label components. It protects from both write-up and write-down.

What are write-up and write down?:

Write-up and write-down apply only to components of type ARRAY and only to write access. Write up occurs when the value protecting data that you are writing to is higher than your value. Write-down is when the value protecting the data is lower than yours. By default neither write-up nor write-down is allowed, meaning that you can only write data that is protected by the same value that you have.

When comparing two values for the same component, which rules are used depends on the type of the component (ARRAY, SET, or TREE) and what type of access is being attempted (read, or write). This table lists the rules, tells when each is used, and describes how the rule determines if access is blocked.

Table 10. Summary of the DB2LBACRULES rules

Rule name	Used when comparing the values of this type of component	Used when attempting this type of access	Access is blocked when this condition is met
DB2LBACREADARRAY	ARRAY	Read	The user's value is lower than the protecting value.
DB2LBACREADSET	SET	Read	There are one or more protecting values that the user does not hold.
DB2LBACREADTREE	TREE	Read	None of the user's values is equal to or an ancestor of one of the protecting values.
DB2LBACWRITEARRAY	ARRAY	Write	The user's value is higher than the protecting value or lower than the protecting value.[1]
DB2LBACWRITESET	SET	Write	There are one or more protecting values that the user does not hold.
DB2LBACWRITETREE	TREE	Write	None of the user's values is equal to or an ancestor of one of the protecting values.

Notes:

1. The DB2LBACWRITEARRAY rule can be thought of as being two different rules combined. One prevents writing to data that is higher than your level (write-up) and the other prevents writing to data that is lower than your level (write-down). When granting an exemption to this rule you can exempt the user from either of these rules or from both.

How the rules handle empty values:

All rules treat empty values the same way. An empty value blocks no other values and is blocked by any non-empty value.

Examples:

DB2LBACREADSET and DB2LBACWRITESET examples:

These examples are valid for a user trying to read or trying to write protected data. They assume that the values are for a component of type SET that has these elements: one two three four

Table 11. Examples of applying the DB2LBACREADSET and DB2LBACWRITESET rules.

User's value	Protecting value	Access blocked?
'one'	'one'	Not blocked. The values are the same.
'(one,two,three)'	'one'	Not blocked. The user's value contains the element 'one'.
'(one,two)'	'(one,two,four)'	Blocked. The element 'four' is in the protecting value but not in the user's value.
'()'	'one'	Blocked. An empty value is blocked by any non-empty value.
'one'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACREADTREE and DB2LBACWRITETREE:

These examples are valid for both read access and write access. They assume that the values are for a component of type TREE that was defined in this way:

```
CREATE SECURITY LABEL COMPONENT mycomp
TREE (
    'Corporate'      ROOT,
    'Publishing'     UNDER 'Corporate',
    'Software'       UNDER 'Corporate',
    'Development'    UNDER 'Software',
    'Sales'          UNDER 'Software',
    'Support'        UNDER 'Software'
    'Business Sales' UNDER 'Sales'
    'Home Sales'     UNDER 'Sales'
)
```

This means the elements are in this arrangement:

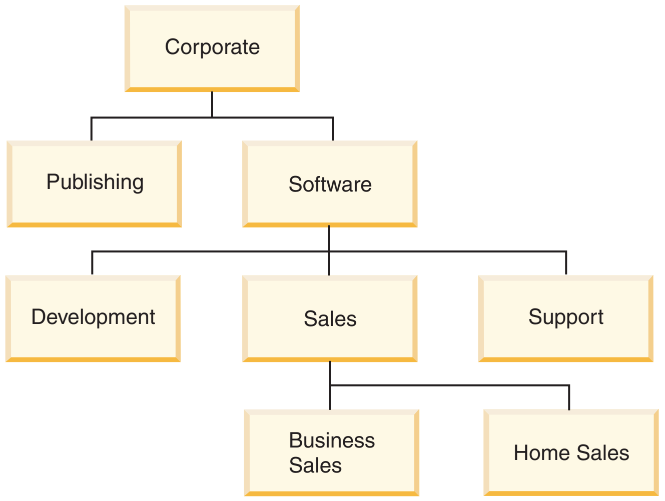


Table 12. Examples of applying the DB2LBACREADTREE and DB2LBACWRITETREE rules.

User's value	Protecting value	Access blocked?
'(Support,Sales)'	'Development'	Blocked. The element 'Development' is not one of the user's values and neither 'Support' nor 'Sales' is an ancestor of 'Development'.
'(Development,Software)'	'(Business Sales,Publishing)'	Not blocked. The element 'Software' is an ancestor of 'Business Sales'.
'(Publishing,Sales)'	'(Publishing,Support)'	Not blocked. The element 'Publishing' is in both sets of values.
'Corporate'	'Development'	Not blocked. The root value is an ancestor of all other values.
'()'	'Sales'	Blocked. An empty value is blocked by any non-empty value.
'Home Sales'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACREADARRAY examples:

These examples are for read access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

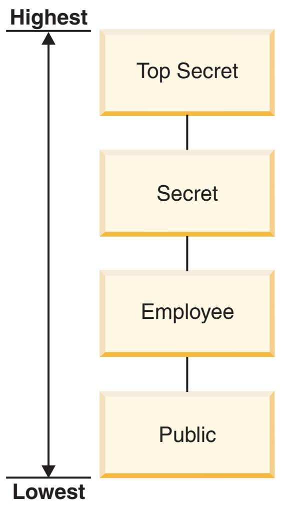


Table 13. Examples of applying the DB2LBACREADARRAY rule.

User's value	Protecting value	Read access blocked?
'Secret'	'Employee'	Not blocked. The element 'Secret' is higher than the element 'Employee'.
'Secret'	'Secret'	Not blocked. The values are the same.

Table 13. Examples of applying the DB2LBACREADARRAY rule. (continued)

User’s value	Protecting value	Read access blocked?
’Secret’	’Top Secret’	Blocked. The element ’Top Secret’ is higher than the element ’Secret’.
'()'	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

DB2LBACWRITEARRAY examples:

These examples are for write access only. They assume that the values are for a component of type ARRAY that includes these elements in this arrangement:

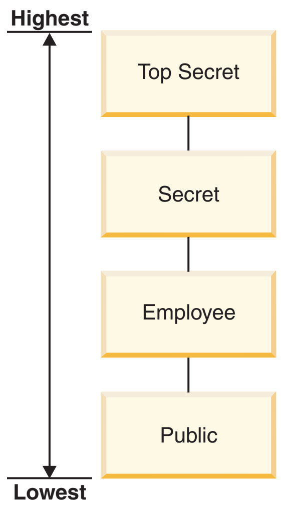


Table 14. Examples of applying the DB2LBACWRITEARRAY rule.

User’s value	Protecting value	Write access blocked?
’Secret’	’Employee’	Blocked. The element ’Employee’ is lower than the element ’Secret’.
’Secret’	’Secret’	Not blocked. The values are the same.
’Secret’	’Top Secret’	Blocked. The element ’Top Secret’ is higher than the element ’Secret’.
'()'	'Public'	Blocked. An empty value is blocked by any non-empty value.
'Public'	'()'	Not blocked. No value is blocked by an empty value.
'()'	'()'	Not blocked. No value is blocked by an empty value.

Related concepts:

- “How LBAC security labels are compared” on page 120

- “LBAC rule sets overview” on page 121

LBAC rule exemptions

Exemptions:

An LBAC rule exemption is part of the label-based access control (LBAC) feature. When you hold an exemption on a particular rule of a particular security policy that rule is not enforced when you try to access data protected by that security policy. An exemption has no effect when comparing security labels of any security policy other than the one for which it was granted.

Example:

> There are two tables: T1 and T2. T1 is protected by security policy P1 and T2 is protected by security policy P2. Both security policies have one component. The component of each is of type ARRAY. T1 and T2 each contain only one row of data. The security label that you hold for read access under security policy P1 does not allow you access to the row in T1. The security label that you hold for read access under security policy P2 does not allow you read access to the row in T2.
>
> Now you are granted an exemption on DB2LBACREADARRAY under P1. You can now read the row from T1 but not the row from T2 because T2 is protected by a different security policy and you do not hold an exemption to the DB2LBACREADARRAY rule in that policy.

You can hold multiple exemptions. If you hold an exemption to every rule used by a security policy then you will have complete access to all data protected by that security policy.

Granting LBAC rule exemptions:

You must have security administrator (SECADM) authority to grant an LBAC rule exemption. To grant an LBAC rule exemption, use the SQL statement GRANT EXEMPTION ON RULE.

When you grant an LBAC rule exemption you provide this information:
- The rule or rules that the exemption is for
- The security policy that the exemption is for
- The user to which you are granting the exemption

Important: LBAC rule exemptions provide very powerful access. Do not grant them without careful consideration.

Revoking LBAC rule exemptions:

You must have security administrator (SECADM) authority to revoke an LBAC rule exemption. To revoke an LBAC rule exemption, use the SQL statement REVOKE EXEMPTION ON RULE.

Related concepts:
- “How LBAC security labels are compared” on page 120
- “LBAC rule sets overview” on page 121

- “LBAC security policies” on page 111

Related reference:

- “GRANT (Exemption) ” on page 1084
- “REVOKE (Exemption) ” on page 1124

Built-in functions for dealing with LBAC security labels

Three built-in functions are provided for dealing with label-based access control (LBAC) security labels. Each is described briefly here and in detail in the *SQL Reference*

SECLABEL:

This built-in function is used to build a security label by specifying a security policy and values for each of the components in the label. The returned value has a data type of DB2SECURITYLABEL and is a security label that is part of the indicated security policy and has the indicated values for the components. It is not necessary that a security label with the indicated values already exists.

Example: Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. If UNCLASSIFIED is an element of the component level, ALPHA and SIGMA are both elements of the component departments, and G2 is an element of the component groups then a security label could be inserted like this:

```
INSERT INTO T1 VALUES ( SECLABEL( 'P1', 'UNCLASSIFIED:(ALPHA,SIGMA):G2' ), 22 )
```

SECLABEL_BY_NAME:

This built-in function accepts the name of a security policy and the name of a security label that is part of that security policy. It then returns the indicated security label as a DB2SECURITYLABEL. You must use this function when inserting an existing security label into a column that has a data type of DB2SECURITYLABEL.

Example: Table T1 has two columns, the first has a data type of DB2SECURITYLABEL and the second has a data type of INTEGER. The security label named L1 is part of security policy P1. This SQL inserts the security label:

```
INSERT INTO T1 VALUES ( SECLABEL_BY_NAME( 'P1', 'L1' ), 22 )
```

This SQL does not work:

```
INSERT INTO T1 VALUES ( P1.L1, 22 )     // Syntax Error!
```

SECLABEL_TO_CHAR:

This built-in function returns a string representation of the values that make up a security label.

Example: Column C1 in table T1 has a data type of DB2SECURITYLABEL. T1 is protected by security policy P1, which has three security label components: level, departments, and groups. There is one row in T1 and the value in column C1 that has these elements for each of the components:

Component	Elements
level	SECRET
departments	DELTA and SIGMA
groups	G3

A user that has LBAC credentials that allow reading the row executes this SQL statement:

```
SELECT SECLABEL_TO_CHAR( 'P1', C1 ) AS C1 FROM T1
```

The output looks like this:

```
C1

'SECRET:(DELTA,SIGMA):G3'
```

Related concepts:
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security labels" on page 118

Related reference:
- "SECLABEL_BY_NAME " on page 1270
- "SECLABEL_TO_CHAR " on page 1271
- "SECLABEL " on page 1270
- "Format for security label values" on page 119

Protection of data using LBAC

Protecting tables:

Label-based access control (LBAC) can be used to protect rows of data, columns of data, or both. Data in a table can only be protected by security labels that are part of the security policy protecting the table. Data protection, including adding a security policy, can be done when creating the table or later by altering the table. You can add a security policy to a table and protect data in that table as part of the same CREATE TABLE or ALTER TABLE statement.

As a general rule you are not allowed to protect data in such a way that your current LBAC credentials do not allow you to write to that data.

Adding a security policy to a table:

You can add a security policy to a table when you create the table by using the SECURITY POLICY clause of the CREATE TABLE statement. You can add a security policy to an existing table by using the ADD SECURITY POLICY clause of the ALTER TABLE statement. You do not need to have SECADM authority or have LBAC credentials to add a security policy to a table.

Security policies cannot be added to types of tables that cannot be protected by LBAC. See the overview of LBAC for a list of table types that cannot be protected by LBAC.

No more than one security policy can be added to any table.

Protecting rows:

You can allow protected rows in a new table by including a column with a data type of DB2SECURITYLABEL when you create the table. The CREATE TABLE statement must also add a security policy to the table. You do not need to have SECADM authority or have any LBAC credentials to create such a table.

You can allow protected rows in an existing table by adding a column that has a data type of DB2SECURITYLABEL. To add such a column, either the table must already be protected by a security policy or the ALTER TABLE statement that adds the column must also add a security policy to the table. When the column is added, the security label you hold for write access is used to protect all existing rows. If you do not hold a security label for write access that is part of the security policy protecting the table then you cannot add a column that has a data type of DB2SECURITYLABEL.

After a table has a column of type DB2SECURITYLABEL you protect each new row of data by storing a security label in that column. The details of how this works are described in the topics about inserting and updating LBAC protected data. You must have LBAC credentials to insert rows into a table that has a column of type DB2SECURITYLABEL.

A column that has a data type of DB2SECURITYLABEL cannot be dropped and cannot be changed to any other data type.

Protecting columns:

You can protect a column when you create the table by using the SECURED WITH column option of the CREATE TABLE statement. You can add protection to an existing column by using the SECURED WITH option in an ALTER TABLE statement.

To protect a column with a particular security label you must have LBAC credentials that allow you to write to data protected by that security label. You do not have to have SECADM authority.

Columns can only be protected by security labels that are part of the security policy protecting the table. You cannot protect columns in a table that has no security policy. You are allowed to protect a table with a security policy and protect one or more columns in the same statement.

You can protect any number of the columns in a table but a column can be protected by no more than one security label.

Related concepts:

- "Inserting of LBAC protected data" on page 133
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security labels" on page 118
- "LBAC security policies" on page 111
- "Removal of LBAC protection from data" on page 142
- "Updating of LBAC protected data" on page 135

Related reference:
- "ALTER TABLE " on page 854
- "CREATE TABLE " on page 956

Reading of LBAC protected data

When you try to read data protected by label-based access control (LBAC), your LBAC credentials for reading are compared to the security label that is protecting the data. If the protecting label does not block your credentials you are allowed to read the data.

In the case of a protected column the protecting security label is defined in the schema of the table. The protecting security label for that column is the same for every row in the table. In the case of a protected row the protecting security label is stored in the row in a column of type DB2SECURITYLABEL. It can be different for every row in the table.

The details of how your LBAC credentials are compared to a security label are given in How LBAC security labels are compared.

Reading protected columns:

When you try to read from a protected column your LBAC credentials are compared with the security label protecting the column. Based on this comparison access will either be blocked or allowed. If access is blocked then an error is returned and the statement fails. Otherwise, the statement proceeds as usual.

Trying to read a column that your LBAC credentials do not allow you to read, causes the entire statement to fail.

Example:

Table T1 has two protected columns. The column C1 is protected by the security label L1. The column C2 is protected by the security label L2.

Assume that user Jyoti has LBAC credentials for reading that allow access to security label L1 but not to L2. If Jyoti issues the following SQL statement, the statement will fail:

```
SELECT * FROM T1
```

The statement fails because column C2 is included in the SELECT clause as part of the wildcard (*).

If Jyoti issues the following SQL statement it will succeed:

```
SELECT C1 FROM T1
```

The only protected column in the SELECT clause is C1, and Jyoti's LBAC credentials allow her to read that column.

Reading protected rows:

If you do not have LBAC credentials that allow you to read a row it is as if that row does not exist for you.

When you read protected rows, only those rows to which your LBAC credentials allow read access are returned. This is true even if the column of type DB2SECURITYLABEL is not part of the SELECT clause.

Depending on their LBAC credentials, different users might see different rows in a table that has protected rows. For example, two users executing the statement `SELECT COUNT(*) FROM T1` may get different results if T1 has protected rows and the users have different LBAC credentials.

Your LBAC credentials affect not only SELECT statements but also other SQL statements like UPDATE, and DELETE. If you do not have LBAC credentials that allow you to read a row, you cannot affect that row.

Example:

Table T1 has these rows and columns. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL.

Table 15.

LASTNAME	DEPTNO	ROWSECURITYLABEL
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3
Bird	55	L2

Assume that user Dan has LBAC credentials that allow him to read data that is protected by security label L1 but not data protected by L2 or L3.

Dan issues the following SQL statement:

```
SELECT * FROM T1
```

The SELECT statement returns only the row for Miller. No error messages or warning are returned.

Dan's view of table T1 is this:

Table 16.

LASTNAME	DEPTNO	ROWSECURITYLABEL
Miller	77	L1

The rows for Rjaibi, Fielding, and Bird are not returned because read access is blocked by their security labels. Dan cannot delete or update these rows. They will also not be included in any aggregate functions. For Dan it is as if those rows do not exist.

Dan issues this SQL statement:

```
SELECT COUNT(*) FROM T1
```

The statement returns a value of 1 because only the row for Miller can be read by the user Dan.

Reading protected rows that contain protected columns:

Column access is checked before row access. If your LBAC credentials for read access are blocked by the security label protecting one of the columns you are selecting then the entire statement fails. If not, the statement continues and only the rows protected by security labels to which your LBAC credentials allow read access are returned.

Example:

The column LASTNAME of table T1 is protected with the security label L1. The column DEPTNO is protected with security label L2. The column ROWSECURITYLABEL has a data type of DB2SECURITYLABEL. T1, including the data, looks like this:

Table 17.

LASTNAME *Protected by L1*	**DEPTNO** *Protected by L2*	**ROWSECURITYLABEL**
Rjaibi	55	L2
Miller	77	L1
Fielding	11	L3

Assume that user Sakari has LBAC credentials that allow reading data protected by security label L1 but not L2 or L3.

Sakari issues this SQL statement:

```
SELECT * FROM T1
```

The statement fails because the SELECT clause uses the wildcard (*) which includes the column DEPTNO. The column DEPTNO is protected by security label L2, which Sakari's LBAC credentials do not allow her to read.

Sakari next issues this SQL statement:

```
SELECT LASTNAME, ROWSECURITYLABEL FROM T1
```

The select clause does not include any columns that Sakari is not able to read so the statement continues. Only one row is returned, however, because each of the other rows is protected by security label L2 or L3.

Table 18.

LASTNAME	**ROWSECURITYLABEL**
Miller	L1

Related concepts:

- “Deleting or dropping of LBAC protected data” on page 139
- “How LBAC security labels are compared” on page 120
- “Inserting of LBAC protected data” on page 133
- “Label-based access control (LBAC) overview” on page 109
- “LBAC security labels” on page 118
- “Updating of LBAC protected data” on page 135

Inserting of LBAC protected data

Inserting to protected columns:

When you try to explicitly insert data to a protected column your LBAC credentials for writing are compared with the security label protecting that column. Based on this comparison access will either be blocked or allowed.

The details of how two security labels are compared are given in How LBAC security labels are compared.

If access is allowed, the statement proceeds as usual. If access is blocked, then the insert fails and an error is returned.

If you are inserting a row but do not provide a value for a protected column then a default value is inserted if one is available. This happens even if your LBAC credentials do not allow write access to that column. A default is available in the following cases:
- The column was declared with the WITH DEFAULT option
- The column is a generated column
- The column has a default value that is given through a BEFORE trigger
- The column has a data type of DB2SECURITYLABEL, in which case security label that you hold for write access is the default value

Inserting protected rows:

When you insert a new row into a table with protected rows you do not have to provide a value for the column that is of type DB2SECURITYLABEL. If you do not provide a value for that column the column is automatically populated with the security label you have been granted for write access. If you have not been granted a security label for write access an error is returned and the insert fails.

By using built-in functions like SECLABEL you can explicitly provide a security label to be inserted in a column of type DB2SECURITYLABEL. The provided security label is only used, however, if your LBAC credentials would allow you to write to data that is protected with the security label you are trying to insert.

If you provide a security label that you would not be able to write to then what happens depends on the security policy that is protecting the table. If the CREATE SECURITY POLICY statement that created the policy included the option RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL then the insert fails and an error is returned. If the CREATE SECURITY POLICY statement did not include the option or if it instead included the OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL option then the security label you provide is ignored and the security label you hold for write access is used instead. No error or warning is issued in this case.

Examples:

Table T1 is protected by a security policy (named P1) that was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option. Table T1 has two columns but no rows. The columns are LASTNAME and LABEL. The column LABEL has a data type of DB2SECURITYLABEL.

User Joe holds a security label L2 for write access. Assume that the security label L2 allows him to write to data protected by security label L2 but not to data protected by security labels L1 or L3.

Joe issues the following SQL statement:

```
INSERT INTO T1 (LASTNAME, DEPTNO) VALUES ('Rjaibi', 11)
```

Because no security label was included in the INSERT statement, Joe's security label for write access is inserted into the LABEL row.

Table T1 now looks like this:

Table 19.

LASTNAME	LABEL
Rjaibi	L2

Joe issues the following SQL statement, in which he explicitly provides the security label to be inserted into the column LABEL:

```
INSERT INTO T1 VALUES ('Miller', SECLABEL_BY_NAME('P1', 'L1') )
```

The SECLABEL_BY_NAME function in the statement returns a security label that is part of security policy P1 and is named L1. Joe is not allowed to write to data that is protected with L1 so he is not allowed to insert L1 into the column LABEL.

Because the security policy protecting T1 was created without the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option the security label that Joe holds for writing is inserted instead. No error or message is returned.

The table now looks like this:

Table 20.

LASTNAME	LABEL
Rjaibi	L2
Miller	L2

If the security policy protecting the table had been created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option then the insert would have failed and an error would have been returned.

Next Joe is granted an exemption to one of the LBAC rules. Assume that his new LBAC credentials allow him to write to data that is protected with security labels L1 and L2. The security label granted to Joe for write access does not change, it is still L2.

Joe issues the following SQL statement:

```
INSERT INTO T1 VALUES ('Bird', SECLABEL_BY_NAME('P1', 'L1') )
```

Because of his new LBAC credentials Joe is able to write to data that is protected by the security label L1. The insertion of L1 is therefore allowed. The table now looks like this:

Table 21.

LASTNAME	LABEL
Rjaibi	L2
Miller	L2
Bird	L1

Related concepts:
- "Deleting or dropping of LBAC protected data" on page 139
- "How LBAC security labels are compared" on page 120
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security labels" on page 118
- "LBAC security policies" on page 111
- "Reading of LBAC protected data" on page 130
- "Updating of LBAC protected data" on page 135

Updating of LBAC protected data

Your LBAC credentials must allow you write access to data before you can update it. In the case of updating a protected row your LBAC credentials must also allow read access to the row.

Updating protected columns:

When you try to update data in a protected column, your LBAC credentials are compared to the security label protecting the column. The comparison made is for write access. If write access is blocked then an error is returned and the statement fails, otherwise the update continues.

The details of how your LBAC credentials are compared to a security label are given in How LBAC security labels are compared.

Example:

Assume there is a table T1 in which column DEPTNO is protected by a security label L2 and column PAYSCALE is protected by a security label L3. T1, including its data, looks like this:

Table 22.

EMPNO	LASTNAME	DEPTNO *Protected by* **L2**	PAYSCALE *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	11	7
3	Bird	11	9

User Lhakpa has no LBAC credentials. He issues this SQL statement:

```
UPDATE T1 SET EMPNO = 4 WHERE LASTNAME = "Bird"
```

This statement executes without error because it does not update any protected columns. T1 now looks like this:

Table 23.

EMPNO	**LASTNAME**	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	11	7
4	Bird	11	9

Lhakpa next issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55 WHERE LASTNAME = "Miller"
```

This statement fails and an error is returned because DEPTNO is protected and Lhakpa has no LBAC credentials.

Assume Lhakpa is granted LBAC credentials and that allow the access summarized in the following table. The details of what those credentials are and what elements are in the security labels are not important for this example.

Security label protecting the data	**Can read?**	**Can Write?**
L2	No	Yes
L3	No	No

Lhakpa issues this SQL statement again:

```
UPDATE T1 SET DEPTNO = 55 WHERE LASTNAME = "Miller"
```

This time the statement executes without error because Lhakpa's LBAC credentials allow him to write to data protected by the security label that is protecting the column DEPTNO. It does not matter that he is not able to read from that same column. The data in T1 now looks like this:

Table 24.

EMPNO	**LASTNAME**	**DEPTNO** *Protected by* **L2**	**PAYSCALE** *Protected by* **L3**
1	Rjaibi	11	4
2	Miller	55	7
4	Bird	11	9

Next Lhakpa issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 55, PAYSCALE = 4 WHERE LASTNAME = "Bird"
```

The column PAYSCALE is protected by the security label L3 and Lhakpa's LBAC credentials do not allow him to write to it. Because Lhakpa is unable to write to the column, the update fails and no data is changed.

Updating protected rows:

If your LBAC credentials do not allow you to read a row then it is as if that row does not exist for you so there is no way for you to update that row. For rows that you are able to read, you must also be able to write to the row in order to update it.

When you try to update a row your LBAC credentials for writing are compared to the security label protecting the row. If write access is blocked the update fails and an error is returned. If write access is not blocked then the update continues.

The update that is performed is done the same way as an update to a non-protected row except for the treatment of the column that has a data type of DB2SECURITYLABEL. If you do not explicitly set the value of that column it is automatically set to the security label that you hold for write access. If you do not have a security label for write access an error is returned and the statement fails.

If the update explicitly sets the column that has a data type of DB2SECURITYLABEL then your LBAC credentials are checked again. If the update you are trying to perform would create a row that your current LBAC credentials would not allow you to write to then an error is returned and the statement fails. Otherwise the column is set to the provided security label.

Example:

Assume that table T1 is protected by a security policy named P1 and has a column named LABEL that has a data type of DB2SECURITYLABEL.

T1, including its data, looks like this:

Table 25.

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1
2	Miller	11	L2
3	Bird	11	L3

Assume that user Jenni has LBAC credentials that allow her to read and write data protected by the security labels L0 and L1 but not data protected by any other security labels. The security label she holds for both read and write is L0. The details of her full credentials and of what elements are in the labels are not important for this example.

Jenni issues this SQL statement:

```
SELECT * FROM T1
```

Jenni sees only one row in the table:

Table 26.

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	11	L1

The rows protected by labels L2 and L3 are not included in the result set because Jenni's LBAC credentials do not allow her to read those rows. For Jenni it is as if those rows do not exist.

Jenni issues these SQL statements:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11;
SELECT * FROM T1;
```

The result set returned by the query looks like this:

Table 27.

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0

The actual data in the table looks like this:

Table 28.

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L0
2	Miller	11	L2
3	Bird	11	L3

The statement executed without error but affected only the first row. The second and third rows are not readable by Jenni so they are not selected for update by the statement even though they meet the condition in the WHERE clause.

Notice that the value of the LABEL column in the updated row has changed even though that column was not explicitly set in the UPDATE statement. The column was set to the security label that Jenni held for writing.

Now Jenni is granted LBAC credentials that allow her to read data protected by any security label. Her LBAC credentials for writing do not change. She is still only able to write to data protected by L0 and L1.

Jenni again issues this SQL statement:

```
UPDATE T1 SET DEPTNO = 44 WHERE DEPTNO = 11
```

This time the update fails because of the second and third rows. Jenni is able to read those rows, so they are selected for update by the statement. She is not, however, able to write to them because they are protected by security labels L2 and L3. The update does not occur and an error is returned.

Jenni now issues this SQL statement:

```
UPDATE T1
 SET DEPTNO = 55, LABEL = SECLABEL_BY_NAME( 'P1', 'L2' )
 WHERE LASTNAME = "Rjaibi"
```

The SECLABEL_BY_NAME function in the statement returns the security label named L2. Jenni is trying to explicitly set the security label protecting the first row. Jenni's LBAC credentials allow her to read the first row, so it is selected for update. Her LBAC credentials allow her to write to rows protected by the security label L0 so she is allowed to update the row. Her LBAC credentials would not, however, allow her to write to a row protected by the security label L2, so she is not allowed to set the column LABEL to that value. The statement fails and an error is returned. No columns in the row are updated.

Jenni now issues this SQL statement:

```
UPDATE T1 SET LABEL = SECLABEL_BY_NAME( 'P1', 'L1' ) WHERE LASTNAME = "Rjaibi"
```

The statement succeeds because she would be able to write to a row protected by the security label L1.

T1 now looks like this:

Table 29.

EMPNO	LASTNAME	DEPTNO	LABEL
1	Rjaibi	44	L1
2	Miller	11	L2
3	Bird	11	L3

Updating protected rows that contain protected columns:

If you try to update protected columns in a table with protected rows then your LBAC credentials must allow writing to of all of the protected columns affected by the update, otherwise the update fails and an error is returned. This is as described in preceding section **Updating protected columns**. If you are allowed to update all of the protected columns affected by the update you will still only be able to update rows that your LBAC credentials allow you to both read from and write to. This is as described in the preceding section **Updating protected rows**. The handling of a column with a data type of DB2SECURITYLABEL is the same whether the update affects protected columns or not.

If the column that has a data type of DB2SECURITYLABEL is itself a protected column then your LBAC credentials must allow you to write to that column or you cannot update any of the rows in the table.

Related concepts:

- "Deleting or dropping of LBAC protected data" on page 139
- "Inserting of LBAC protected data" on page 133
- "Reading of LBAC protected data" on page 130

Deleting or dropping of LBAC protected data

Deleting protected rows:

If your LBAC credentials do not allow you to read a row then it is as if that row does not exist for you so there is no way for you to delete it.

To delete a row that you are able to read, your LBAC credentials must also allow you to write to the row. When you try to delete a row, your LBAC credentials for writing are compared to the security label protecting the row. If the protecting security label blocks write access by your LBAC credentials, the DELETE statement fails, an error is returned, and no rows are deleted.

Example:

Protected table T1 has these rows:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Pat has LBAC credentials such that her access is as summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of her LBAC credentials and of the security labels are unimportant for this example.

Pat issues the following SQL statement:

```
SELECT * FROM T1 WHERE DEPTNO != 999
```

The statement executes and returns this result set:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2

The last row of T1 is not included in the results because Pat does not have read access to that row. It is as if that row does not exist for Pat.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO != 999
```

Pat does not have write access to the first or third row, both of which are protected by L2. So even though she can read the rows she cannot delete them. The DELETE statement fails and no rows are deleted.

Pat issues this SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77;
```

This statement succeeds because Pat is able to write to the row with Miller in the LASTNAME column. That is the only row selected by the statement. The row with Fielding in the LASTNAME column is not selected because Pat's LBAC credentials do not allow her to read that row. That row is never considered for the delete so no error occurs.

The actual rows of the table now look like this:

LASTNAME	DEPTNO	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

Deleting rows that have protected columns:

To delete any row in a table that has protected columns you must have LBAC credentials that allow you to write to all protected columns in the table. If there is any row in the table that your LBAC credentials do not allow you to write to then the delete will fail and an error will be returned.

If the table has both protected columns and protected rows then to delete a particular row you must have LBAC credentials that allow you to write to every protected column in the table and also to read from and write to the row that you want to delete.

*Example***:**

In protected table T1, the column DEPTNO is protected by the security label L2. T1 contains these rows:

LASTNAME	DEPTNO *Protected by* **L2**	LABEL
Rjaibi	55	L2
Miller	77	L1
Bird	55	L2
Fielding	77	L3

Assume that user Benny has LBAC credentials that allow him the access summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	No
L3	No	No

The exact details of his LBAC credentials and of the security labels are unimportant for this example.

Benny issues the following SQL statement:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

The statement fails because Benny does not have write access to the column DEPTNO.

Now Benny's LBAC credentials are changed so that he has access as summarized in this table:

Security label	Read access?	Write access?
L1	Yes	Yes
L2	Yes	Yes
L3	Yes	No

Benny issues this SQL statement again:

```
DELETE FROM T1 WHERE DEPTNO = 77
```

This time Benny has write access to the column DEPTNO so the delete continues. The delete statement selects only the row that has a value of Miller in the LASTNAME column. The row that has a value of Fielding in the LASTNAME column is not selected because Benny's LBAC credentials do not allow him to read that row. Because the row is not selected for deletion by the statement it does not matter that Benny is unable to write to the row.

The one row selected is protected by the security label L1. Benny's LBAC credentials allow him to write to data protected by L1 so the delete is successful.

The actual rows in table T1 now look like this:

LASTNAME	DEPTNO *Protected by* **L2**	LABEL
Rjaibi	55	L2
Bird	55	L2
Fielding	77	L3

Dropping protected data:

You cannot drop a column that is protected by a security label unless your LBAC credentials allow you to write to that column.

A column with a data type of DB2SECURITYLABEL cannot be dropped from a table. To remove it you must first drop the security policy from the table. When you drop the security policy the table is no longer protected with LBAC and the data type of the column is automatically changed from DB2SECURITYLABEL to VARCHAR(128) FOR BIT DATA. The column can then be dropped.

Your LBAC credentials do not prevent you from dropping entire tables or databases that contain protected data. If you would normally have permission to drop a table or a database you do not need any LBAC credentials to do so, even if the database contains protected data.

Related concepts:
- “Inserting of LBAC protected data” on page 133
- “Reading of LBAC protected data” on page 130
- “Updating of LBAC protected data” on page 135

Removal of LBAC protection from data

Removing a security policy from a table:

You must have SECADM authority to remove the security policy from a table. To remove the security policy from a table you use the DROP SECURITY POLICY clause of the ALTER TABLE statement. This also automatically removes protection from all rows and all columns of the table.

Removing protection from rows:

In a table that has protected rows every row must be protected by a security label. There is no way to remove LBAC protection from individual rows.

A column of type DB2SECURITYLABEL cannot be altered or removed except by removing the security policy from the table.

Removing protection from columns:

Protection of a column can be removed using the DROP COLUMN SECURITY clause of the SQL statement ALTER TABLE. To remove the protection from a column you must have LBAC credentials that allow you to read from and write to that column in addition to the normal privileges and authorities needed to alter a table.

Related concepts:
- "Label-based access control (LBAC) overview" on page 109
- "LBAC security labels" on page 118
- "LBAC security policies" on page 111
- "Protection of data using LBAC" on page 128

Related reference:
- "ALTER TABLE " on page 854
- "CREATE TABLE " on page 956

Chapter 15. Security considerations

1. Gaining Access to Data through Indirect Means

   The following are indirect means through which users can gain access to data they might not be authorized for:

   - **Catalog views:** The DB2 database system catalog views store metadata and statistics about database objects. Users with SELECT access to the catalog views can gain some knowledge about data that they might not be qualified for. For better security, make sure that only qualified users have access to the catalog views.

     Note: In DB2 Universal Database™ V8 or earlier, SELECT access on the catalog views was granted to PUBLIC by default. In DB2 V9.1 database systems, users can choose whether SELECT access to the catalog views is granted to PUBLIC or not by using the new RESTRICTIVE option on the CREATE DATABASE command.
   - **Visual explain:** Visual explain shows the access plan chosen by the query optimizer for a particular query. The visual explain information also includes statistics about columns referenced in the query. These statistics can reveal information about a table's contents.
   - **Explain snapshot:** The explain snapshot is compressed information that is collected when an SQL or XQuery statement is explained. It is stored as a binary large object (BLOB) in the EXPLAIN_STATEMENT table, and contains column statistics that can reveal information about table data. For better security, access to the explain tables should be granted to qualified users only.
   - **Log reader functions:** A user authorized to run a function that reads the logs can gain access to data they might not be authorized for if they are able to understand the format of a log record. These functions read the logs:

Function	**Authority needed in order to execute the function**
db2ReadLog	SYSADM or DBADM
db2ReadLogNoConn	None.

   - **Replication:** When you replicate data, even the protected data is reproduced at the target location. For better security, make sure that the target location is at least as secure as the source location.
   - **Exception tables:** When you specify an exception table while loading data into a table, users with access to the exception table can gain information that they might not be authorized for. For better security, only grant access to the exception table to authorized users and drop the exception table as soon as you are done with it.
   - **Backup table space or database:** Users with the authority to run the backup command can take a backup of a database or a table space, including any protected data, and restore the data somewhere else. The backup can include data that the user might not otherwise have access to.

     The backup command can be executed by users with SYSADM, SYSCTRL, or SYSMAINT authority.
   - **Set session authorization:** In DB2 Universal Database V8 or earlier a user with DBADM authority could use the SET SESSION AUTHORIZATION SQL statement to set the session authorization ID to any database user. In DB2

V9.1 database systems a user must be explicitly authorized through the GRANT SETSESSIONUSER statement before they can set the session authorization ID.

When migrating an existing database to a DB2 V9.1 database system, however, a user with existing explicit DBADM authority (for example. granted in SYSCAT.DBAUTH) will keep the ability to set the session authorization to any database user. This is allowed so that existing applications will continue to work. Being able to set the session authorization potentially allows access to all protected data. For more restrictive security, you can override this setting by executing the REVOKE SETSESSIONUSER SQL statement.

- **Statement and deadlock monitoring:** As part of the deadlock monitoring activity of DB2 database management systems, values associated with parameter markers are written to the monitoring output when the WITH VALUES clause is specified. A user with access to the monitoring output can gain access to information for which they might not be authorized.
- **Traces:** A trace can contain table data. A user with access to such a trace can gain access to information that they might not be authorized for.
- **Dump files:** To help in debugging certain problems, DB2 database products might generate memory dump files in the `sqllib\db2dump` directory. These memory dump files might contain table data. If they do, users with access to the files can gain access to information that they might not be authorized for. For better security you should limit access to the `sqllib\db2dump` directory.
- **db2dart:** The db2dart tool examines a database and reports any architectural errors that it finds. The tool can access table data and DB2 does not enforce access control for that access. A user with the authority to run the db2dart tool or with access to the db2dart output can gain access to information that they might not be authorized for.
- **REOPT bind option:** When the REOPT bind option is specified, explain snapshot information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. The explain will also show input data values.
- **db2cat:** The db2cat tool is used to dump a table’s packed descriptor. The table’s packed descriptor contains statistics that can reveal information about a table's contents. A user who runs the db2cat tool or has access to the output can gain access to information that they might not be authorized for.

2. Default Privileges Granted Upon Creating a Database

The following are the default privileges to certain system tables granted when a database is created:

1. SYSIBM.SYSDBAUTH
   - The database creator is granted the following privileges:
     - DBADM
     - CREATETAB
     - CREATEROLE
     - BINDADD
     - CONNECT
     - NOFENCE
     - IMPLSCHEMA
     - LOAD
     - EXTERNALROUTINE

    - QUIESCECONNECT
- The special group PUBLIC is granted the following privileges:
    - CREATETAB
    - BINDADD
    - CONNECT
    - IMPLSCHEMA
2. SYSIBM.SYSTABAUTH
    - The special group PUBLIC is granted the following privileges:
        - SELECT on all SYSCAT and SYSIBM tables
        - SELECT and UPDATE on all SYSSTAT tables
3. SYSIBM.SYSROUTINEAUTH
    - The special group PUBLIC is granted the following privileges:
        - EXECUTE with GRANT on all procedures in schema
        - SQLJ EXECUTE with GRANT on all functions and procedures in schema SYSFUN
        - EXECUTE with GRANT on all functions and procedures in schema SYSPROC
        - EXECUTE on all table functions in schema SYSIBM
        - EXECUTE on all other procedures in schema SYSIBM
4. SYSIBM.SYSPACKAGEAUTH
    - The database creator is granted the following privileges:
        - CONTROL on all packages created in the NULLID schema
        - BIND with GRANT on all packages created in the NULLID schema
        - EXECUTE with GRANT on all packages created in the NULLID schema
        -
    - The special group PUBLIC is granted the following privileges:
        - BIND on all packages created in the NULLID schema
        - EXECUTE on all packages created in the NULLID schema
5. SYSIBM.SCHEMAAUTH
    - The special group PUBLIC is granted the following privileges:
        - CREATEIN on schema SQLJ
        - CREATE IN on schema NULLID
6. SYSIBM.TBSPACEAUTH
    - The special group PUBLIC is granted the following privileges:
        - USE on table space USERSPACE1

Related concepts:
- "Explain snapshot" in *Administration Guide: Implementation*
- "Visual Explain tool" in *Administration Guide: Implementation*

Chapter 16. Authorization ID privileges

Authorization ID privileges involve actions on authorization IDs. There is currently only one such privilege: the SETSESSIONUSER privilege.

The SETSESSIONUSER privilege can be granted to a user or to a group and allows the holder to switch identities to any of the authorization IDs on which the privilege was granted. The identity switch is made by using the SQL statement SET SESSION AUTHORIZATION. The SETSESSIONUSER privilege can only be granted by a user holding SECADM authority.

Note: When you migrate a DB2 UDB database to DB2 Version 9.1, authorization IDs with explicit DBADM authority on that database will automatically be granted SETSESSIONUSER privilege on PUBLIC. This prevents breaking applications that rely on authorization IDs with DBADM authority being able to set the session authorization ID to any authorization ID. This does not happen when the authorization ID has SYSADM authority but has not been explicitly granted DBADM.

Related concepts:

- "Authorization, privileges, and object ownership" on page 63

Related reference:

- "SET SESSION AUTHORIZATION statement" in *SQL Reference, Volume 2*

Chapter 17. User responsibilities for security

In DB2, access to the database and its resources are typically controlled by the system administrator (SYSADM) or database administrator (DBADM) through the use facilities such as the GRANT and REVOKE statements. Database users also have an important role in maintaining the security of the database, and the objects and data in the database. At a minimum, you should implement the following guidelines to protect against unauthorized access to your workstation, the database, and data within the database:

- Lock your workstation when you are away from your desk.
- Ensure that you use a user ID and a password to log on to your computer and to log on to DB2.

  If you are working on an operating system that allows for NULL passwords (that is, you do not have to supply a password to log on to your machine), explicitly define a password for your user ID.

  Passwords are ordinarily a minimum of 8 characters, and should be changed regularly. If you are not certain about the practices followed at your site, check your site's security regulations.
- Do not give your password to unauthorized users.

  Some sites require that you provide your manager with your password. If you are not certain about the practices followed at your site, check your site's security regulations.
- If an unexpected event occurs, or you suspect that an unauthorized individual has accessed either your workstation or DB2, contact your manager or database administrator immediately.
- When using the CONNECT statement to connect to the database from the command line processor (CLP), allow DB2 to prompt you for the password, rather than entering it with the CONNECT statement. This practice prevents the password from being entered into the command history of the operating system.
- Do not hardcode passwords into applications that connect to the database.

Chapter 18. Extended Windows security using DB2ADMNS and DB2USERS groups

For the server version of the DB2 database manager, extended security is implicitly *enabled* by default. However, for the client version, extended security is implicitly *disabled* by default; you *must* explicitly select extended security during installation to have it enabled. During DB2 installation on a client, select the Enable operating system security check box on the Enable operating system security for DB2 object panel. the installer creates two new groups, DB2ADMNS and DB2USERS. DB2ADMNS and DB2USERS are the default group names; optionally, you can specify different names for these groups at installation time (if you select silent install, you can change these names within the install response file). If you choose to use groups that already exist on your system, be aware that the privileges of these groups will be modified. They will be given the privileges, as required, listed in the table, below. It is important to understand that these groups are used for protection at the *operating-system* level and are in no way associated with DB2 authority levels, such as SYSADM, SYSMAINT, and SYSCTRL. However, instead of using the default Administrator's group, your database administrator can use the DB2ADMNS group for one or all of the DB2 authority levels, at the discretion of the installer or administrator. It is recommended that if you are specifying a SYSADM group, then that should be the DB2ADMNS group. This can be established during installation or subsequently, by an administrator.

The DB2ADMNS and DB2USERS groups provide members with the following abilities:

- DB2ADMNS

  Full control over all DB2 objects (see the list of protected objects, below)
- DB2USERS

  Read and Execute access for all DB2 objects located in the installation and instance directories, but no access to objects under the database system directory and limited access to IPC resources

  For certain objects, there may be additional privileges available, as required (for example, write privileges, add or update file privileges, and so on). Members of this group have no access to objects under the database system directory.

  Note: The meaning of Execute access depends on the object; for example, for a **.dll** or **.exe** file having Execute access means you have authority to execute the file, however, for a directory it means you have authority to traverse the directory.

It is recommended that all DB2 administrators be members of the DB2ADMNS group (as well as being members of the local Administrators group), but this is not a strict requirement. Everyone else who requires access to the DB2 database system *must* be a member of the DB2USERS group. To add a user to one of these groups:

1. Launch the Users and Passwords Manager tool.
2. Select the user name to add from the list.
3. Click Properties. In the Properties window, click the Group membership tab.
4. Select the Other radio button.
5. Select the appropriate group from the drop-down list.

Adding extended security after installation (db2extsec command):

If the DB2 database system was installed without extended security enabled, you can enable it by executing the command **db2extsec** (called **db2secv82** in earlier releases). To execute the **db2extsec**command you must be a member of the local Administrators group so that you have the authority to modify the ACL of the protected objects.

You can run the **db2extsec** command multiple times, if necessary, however, if this is done, you cannot disable extended security unless you issue the **db2extsec –r** command immediately after *each* execution of **db2extsec**.

Removing extended security:

CAUTION:
It is not recommend to remove extended security once it has been enabled.

You can remove extended security by running the command **db2extsec -r**, however, this will only succeed if no other database operations (such as creating a database, creating a new instance, adding tablespaces, and so on) have been performed after enabling extended security. The safest way to remove the extended security option is to uninstall the DB2 database system, delete all the relevant DB2 directories (including the database directories) and then reinstall the DB2 database system without extended security enabled.

Protected objects:

The *static* objects that can be protected using the DB2ADMNS and DB2USERS groups are:
- File system
  - File
  - Directory
- Services
- Registry keys

The *dynamic* objects that can be protected using the DB2ADMNS and DB2USERS groups are:
- IPC resources, including:
  - Pipes
  - Semaphores
  - Events
- Shared memory

Privileges owned by the DB2ADMNS and DB2USERS groups:

The privileges assigned to the DB2ADMNS and DB2USERS groups are listed in the following table:

Table 30. Privileges for DB2ADMNS and DB2USERS groups

Privilege	DB2ADMNS	DB2USERS	Reason
Create a token object (SeCreateTokenPrivilege)	Y	N	Token manipulation (required for certain token manipulation operations and used in authentication and authorization)

Table 30. Privileges for DB2ADMNS and DB2USERS groups (continued)

Privilege	DB2ADMNS	DB2USERS	Reason
Replace a process level token (SeAssignPrimaryTokenPrivilege)	Y	N	Create process as another user
Increase quotas (SeIncreaseQuotaPrivilege)	Y	N	Create process as another user
Act as part of the operating system (SeTcbPrivilege)	Y	N	LogonUser (required prior to Windows XP in order to execute the LogonUser API for authentication purposes)
Generate security audits (SeSecurityPrivilege)	Y	N	Manipulate audit and security log
Take ownership of files or other objects (SeTakeOwnershipPrivilege)	Y	N	Modify object ACLs
Increase scheduling priority (SeIncreaseBasePriorityPrivilege)	Y	N	Modify our process working set
Backup files and directories (SeBackupPrivilege)	Y	N	Profile/Registry manipulation (required to perform certain user profile and registry manipulation routines: LoadUserProfile, RegSaveKey(Ex), RegRestoreKey, RegReplaceKey, RegLoadKey(Ex))
Restore files and directories (SeRestorePrivilege)	Y	N	Profile/Registry manipulation (required to perform certain user profile and registry manipulation routines: LoadUserProfile, RegSaveKey(Ex), RegRestoreKey, RegReplaceKey, RegLoadKey(Ex))
Debug programs (SeDebugPrivilege)	Y	N	Token manipulation (required for certain token manipulation operations and used in authentication and authorization)
Manage auditing and security log (SeAuditPrivilege)	Y	N	Generate auditing log entries
Log on as a service (SeServiceLogonRight)	Y	N	Run DB2 as a service
Access this computer from the network (SeNetworkLogonRight)	Y	Y	Allow network credentials (allows the DB2 database manager to use the LOGON32_LOGON_NETWORK option to authenticate, which has performance implications)
Impersonate a client after authentication (SeImpersonatePrivilege)	Y	N	Client impersonation (required for Windowsto allow use of certain APIs to impersonate DB2 clients: ImpersonateLoggedOnUser, ImpersonateSelf, RevertToSelf, and so on)
Lock pages in memory (SeLockMemoryPrivilege)	Y	N	AWE/Large Page support
Create global objects (SeCreateGlobalPrivilege)	Y	Y	Terminal Server support (required on Windows)

Related tasks:

- "Adding your user ID to the DB2ADMNS and DB2USERS user groups (Windows)" in *Quick Beginnings for DB2 Servers*

Related reference:

- "Required user accounts for installation of DB2 server products (Windows)" in *Quick Beginnings for DB2 Servers*
- "db2extsec - Set permissions for DB2 objects" on page 490

Chapter 19. Firewall considerations

Introduction to firewall support

A *firewall* is a set of related programs, located at a network gateway server, that are used to prevent unauthorized access to a system or network.

There are four types of firewalls:
1. Network level, packet-filter, or screening router firewalls
2. Classical application level proxy firewalls
3. Circuit level or transparent proxy firewalls
4. Stateful multi-layer inspection (SMLI) firewalls

There are existing firewall products that incorporate one of the firewall types listed above. There are many other firewall products that incorporate some combination of the above types.

Related concepts:
- "Application proxy firewalls" on page 158
- "Circuit level firewalls" on page 158
- "Screening router firewalls" on page 157
- "Stateful multi-layer inspection (SMLI) firewalls" on page 158

Screening router firewalls

This type of firewall is also known as a network level or packet-filter firewall. Such a firewall works by screening incoming packets by protocol attributes. The protocol attributes screened may include source or destination address, type of protocol, source or destination port, or some other protocol-specific attributes.

For all firewall solutions (except SOCKS), you need to ensure that all the ports used by DB2 database are open for incoming and outgoing packets. DB2 database uses port 523 for the DB2 Administration Server (DAS), which is used by the DB2 database tools. Determine the ports used by all your server instances by using the services file to map the service name in the server database manager configuration file to its port number.

Related concepts:
- "Introduction to firewall support" on page 157

Application proxy firewalls

A proxy or proxy server is a technique that acts as an intermediary between a Web client and a Web server. A proxy firewall acts as a gateway for requests arriving from clients. When client requests are received at the firewall, the final server destination address is determined by the proxy software. The application proxy translates the address, performs additional access control checking and logging as necessary, and connects to the server on behalf of the client.

The DB2 Connect product on a firewall machine can act as a proxy to the destination server. Also, a DB2 database server on the firewall, acting as a hop server to the final destination server, acts like an application proxy.

Related concepts:

- "Introduction to firewall support" on page 157

Circuit level firewalls

This type of firewall is also known as a transparent proxy firewall. A transparent proxy firewall does not modify the request or response beyond what is required for proxy authentication and identification. An example of a transparent proxy firewall is SOCKS.

DB2 database supports SOCKS Version 4.

Related concepts:

- "Introduction to firewall support" on page 157

Stateful multi-layer inspection (SMLI) firewalls

This type of firewall is a sophisticated form of packet-filtering that examines all seven layers of the Open System Interconnection (OSI) model. Each packet is examined and compared against known states of friendly packets. While screening router firewalls only examine the packet header, SMLI firewalls examine the entire packet including the data.

Related concepts:

- "Introduction to firewall support" on page 157

Part 4. System catalogs and security maintenance

Chapter 20. System catalogs and security maintenance

Catalog views

The database manager maintains a set of base tables and views that contain information about the data under its control. These base tables and views are collectively known as the *catalog*. The catalog contains information about the logical and physical structure of database objects such as tables, views, indexes, packages, and functions. It also contains statistical information. The database manager ensures that the descriptions in the catalog are always accurate.

The catalog views are like any other database view. SQL statements can be used to look at the data in the catalog views. A set of updatable catalog views can be used to modify certain values in the catalog.

Related reference:

- "System catalog views" on page 168

Using the system catalog for security issues

Information about each database is automatically maintained in a set of views called the system catalog, which is created when the database is created. This system catalog describes tables, columns, indexes, programs, privileges, and other objects.

The following views and table functions list information about privileges held by users, identities of users granting privileges, and object ownership:

SYSCAT.DBAUTH	Lists the database privileges
SYSCAT.TABAUTH	Lists the table and view privileges
SYSCAT.COLAUTH	Lists the column privileges
SYSCAT.PACKAGEAUTH	Lists the package privileges

SYSCAT.INDEXAUTH	Lists the index privileges
SYSCAT.SCHEMAAUTH	Lists the schema privileges
SYSCAT.PASSTHRUAUTH	Lists the server privilege
SYSCAT.ROUTINEAUTH	Lists the routine (functions, methods, and stored procedures) privileges
SYSCAT.SURROGATEAUTHIDS	Lists the authorization IDs for which another authorization ID can act as a surrogate.

Privileges granted to users by the system will have SYSIBM as the grantor. SYSADM, SYSMAINT SYSCTRL, and SYSMON are not listed in the system catalog.

The CREATE and GRANT statements place privileges in the system catalog. Users with SYSADM and DBADM authorities can grant and revoke SELECT privilege on the system catalog views.

Related tasks:
- "Retrieving all names with DBADM authority" on page 163
- "Retrieving all privileges granted to users" on page 165
- "Retrieving authorization names with granted privileges" on page 162
- "Retrieving names authorized to access a table" on page 164
- "Securing the system catalog view" on page 166

Related reference:
- "SYSCAT.COLAUTH " on page 173
- "SYSCAT.DBAUTH " on page 173
- "SYSCAT.INDEXAUTH " on page 174
- "SYSCAT.PACKAGEAUTH " on page 175
- "SYSCAT.PASSTHRUAUTH " on page 176
- "SYSCAT.ROUTINEAUTH " on page 177
- "SYSCAT.SCHEMAAUTH " on page 177
- "SYSCAT.TABAUTH " on page 193

Details on using the system catalog for security issues

This section reviews some of the ways to determine who has what privileges within the database.

Retrieving authorization names with granted privileges

Procedure:

Starting with version 9.1 of the DB2 database manager, you can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database. For example, the following query retrieves all explicit privileges and the authorization IDs to which they were granted, plus other information, from the PRIVILEGES administrative view:

```
SELECT AUTHID, PRIVILEGE, OBJECTNAME, OBJECTSCHEMA, OBJECTTYPE FROM SYSIBMADM.PRIVILEGES
```

The following query uses the AUTHORIZATIONIDS administrative view to find all the authorization IDs that have been granted privileges or authorities, and to show their types:

```
SELECT AUTHID, AUTHIDTYPE FROM SYSIBMADM.AUTHORIZATIONIDS
```

You can also use the SYSIBMADM.OBJECTOWNERS administrative view and the SYSPROC.AUTH_LIST_GROUPS_FOR_AUTHID table function to find security-related information.

Prior to version 9.1, no single system catalog view contained information about all privileges. For releases earlier than version 9.1, the following statement retrieves all authorization names with privileges:

```
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'DATABASE' FROM SYSCAT.DBAUTH
   UNION
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'TABLE   ' FROM SYSCAT.TABAUTH
   UNION
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'PACKAGE ' FROM SYSCAT.PACKAGEAUTH
   UNION
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'INDEX   ' FROM SYSCAT.INDEXAUTH
   UNION
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'COLUMN  ' FROM SYSCAT.COLAUTH
   UNION
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SCHEMA  ' FROM SYSCAT.SCHEMAAUTH
   UNION
   SELECT DISTINCT GRANTEE, GRANTEETYPE, 'SERVER  ' FROM SYSCAT.PASSTHRUAUTH
   ORDER BY GRANTEE, GRANTEETYPE, 3
```

Periodically, the list retrieved by this statement should be compared with lists of user and group names defined in the system security facility. You can then identify those authorization names that are no longer valid.

Note: If you are supporting remote database clients, it is possible that the authorization name is defined at the remote client only and not on your database server machine.

Related concepts:

- "Using the system catalog for security issues" on page 161

Related reference:

- "AUTH_LIST_GROUPS_FOR_AUTHID table function ÔÇô Retrieve group membership list for a given authorization ID" in *Administrative SQL Routines and Views*
- "AUTHORIZATIONIDS administrative view ÔÇô Retrieve authorization IDs and types" in *Administrative SQL Routines and Views*
- "OBJECTOWNERS administrative view ÔÇô Retrieve object ownership information" in *Administrative SQL Routines and Views*
- "PRIVILEGES administrative view ÔÇô Retrieve privilege information" in *Administrative SQL Routines and Views*

Retrieving all names with DBADM authority

Procedure:

The following statement retrieves all authorization names that have been directly granted DBADM authority:

```
SELECT DISTINCT GRANTEE, GRANTEETYPE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
```

Note: This query will not return information about authorization names that acquired DBADM authority implicitly by having SYSADM authority.

Related concepts:
- "Database administration authority (DBADM)" on page 76
- "Using the system catalog for security issues" on page 161

Retrieving names authorized to access a table

Procedure:

Starting with version 9.1 of the DB2 database manager, you can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database. The following statement retrieves all authorization names (and their types) that are directly authorized to access the table EMPLOYEE with the qualifier JAMES:

```
SELECT DISTINCT AUTHID, AUTHIDTYPE FROM SYSIBMADM.PRIVILEGES
    WHERE OBJECTNAME = 'EMPLOYEE' AND OBJECTSCHEMA = 'JAMES'
```

For releases earlier than version 9.1, the following query retrieves the same information:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
   WHERE TABNAME = 'EMPLOYEE'
     AND TABSCHEMA = 'JAMES'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
   WHERE TABNAME = 'EMPLOYEE'
     AND TABSCHEMA = 'JAMES'
```

To find out who can update the table EMPLOYEE with the qualifier JAMES, issue the following statement:

```
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.TABAUTH
   WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
      (CONTROLAUTH  = 'Y' OR
       UPDATEAUTH IN ('G','Y'))
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.DBAUTH
   WHERE DBADMAUTH = 'Y'
UNION
SELECT DISTINCT GRANTEETYPE, GRANTEE FROM SYSCAT.COLAUTH
   WHERE TABNAME = 'EMPLOYEE' AND TABSCHEMA = 'JAMES' AND
   PRIVTYPE = 'U'
```

This retrieves any authorization names with DBADM authority, as well as those names to which CONTROL or UPDATE privileges have been directly granted. However, it will not return the authorization names of users who only hold SYSADM authority.

Remember that some of the authorization names may be groups, not just individual users.

Related concepts:
- "Table and view privileges" on page 79
- "Using the system catalog for security issues" on page 161

Related reference:

- "PRIVILEGES administrative view ÔÇô Retrieve privilege information" in *Administrative SQL Routines and Views*

Retrieving all privileges granted to users

Procedure:

By making queries on the system catalog views, users can retrieve a list of the privileges they hold and a list of the privileges they have granted to other users. Starting with version 9.1 of the DB2 database manager, you can use the PRIVILEGES and other administrative views to retrieve information about the authorization names that have been granted privileges in a database. For example, the following query retrieves all the privileges granted to the current session authorization ID:

```
SELECT * FROM SYSIBMADM.PRIVILEGES
 WHERE AUTHID = SESSION_USER AND AUTHIDTYPE = 'U'
```

The keyword SESSION_USER in this statement is a special register that is equal to the value of the current user's authorization name.

For releases earlier than version 9.1, the following examples provide similar information. For example, the following statement retrieves a list of the database privileges that have been directly granted to the individual authorization name JAMES:

```
   SELECT * FROM SYSCAT.DBAUTH
      WHERE GRANTEE = 'JAMES' AND GRANTEETYPE = 'U'
```

The following statement retrieves a list of the table privileges that were directly granted by the user JAMES:

```
   SELECT * FROM SYSCAT.TABAUTH
      WHERE GRANTOR  = 'JAMES'
```

The following statement retrieves a list of the individual column privileges that were directly granted by the user JAMES:

```
   SELECT * FROM SYSCAT.COLAUTH
      WHERE GRANTOR  = 'JAMES'
```

Related concepts:

- "Database authorities" on page 74
- "Authorization, privileges, and object ownership" on page 63
- "Using the system catalog for security issues" on page 161

Related tasks:

- "Revoking privileges" on page 97
- "Granting privileges" on page 95

Related reference:

- "PRIVILEGES administrative view ÔÇô Retrieve privilege information" in *Administrative SQL Routines and Views*

Securing the system catalog view

As the system catalog views describe every object in the database, if you have sensitive data, you might want to restrict their access. Starting with version 9.1 of the DB2 database manager, you can use the CREATE DATABASE ... RESTRICTIVE command to create a database in which no privileges are automatically granted to PUBLIC. In this case, none of the following normal default grant actions occur:

- CREATETAB
- BINDADD
- CONNECT
- IMPLSCHEMA
- EXECUTE with GRANT on all procedures in schema SQLJ
- EXECUTE with GRANT on all functions and procedures in schema SYSPROC
- BIND on all packages created in the NULLID schema
- EXECUTE on all packages created in the NULLID schema
- CREATEIN on schema SQLJ
- CREATEIN on schema NULLID
- USE on table space USERSPACE1
- SELECT access to the SYSIBM catalog tables
- SELECT access to the SYSCAT catalog views
- SELECT access to the SYSIBMADM administrative views
- SELECT access to the SYSSTAT catalog views
- UPDATE access to the SYSSTAT catalog views

If you have created a database using the RESTRICTIVE option, and you want to check that the permissions granted to PUBLIC are limited, you can issue the following query to verify which schemas PUBLIC can access:

```
SELECT DISTINCT OBJECTSCHEMA FROM SYSIBMADM.PRIVILEGES WHERE AUTHID='PUBLIC'
```

```
OBJECTSCHEMA
------------
SYSFUN
SYSIBM
SYSPROC
```

To see what access PUBLIC still has to SYSIBM, you can issue the following query to check what privileges are granted on SYSIBM. The results show that only EXECUTE on certain procedures and functions is granted.

```
SELECT * FROM SYSIBMADM.PRIVILEGES WHERE OBJECTSCHEMA = 'SYSIBM'
```

```
AUTHID       AUTHIDTYPE  PRIVILEGE   GRANTABLE  OBJECTNAME         OBJECTSCHEMA    OBJECTTYPE
---------... ----------  ----------  ---------  ---------------... ------------... ----------
PUBLIC       G           EXECUTE     N          SQL060207192129400 SYSPROC         FUNCTION
PUBLIC       G           EXECUTE     N          SQL060207192129700 SYSPROC         FUNCTION
PUBLIC       G           EXECUTE     N          SQL060207192129701 SYSPROC
...
PUBLIC       G           EXECUTE     Y          TABLES             SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          TABLEPRIVILEGES    SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          STATISTICS         SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          SPECIALCOLUMNS     SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          PROCEDURES         SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          PROCEDURECOLS      SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          PRIMARYKEYS        SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          FOREIGNKEYS        SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          COLUMNS            SYSIBM          PROCEDURE
PUBLIC       G           EXECUTE     Y          COLPRIVILEGES      SYSIBM          PROCEDURE
PUBLIC      G           EXECUTE     Y           UDTS               SYSIBM        PROCEDURE
PUBLIC      G           EXECUTE     Y           GETTYPEINFO        SYSIBM        PROCEDURE
PUBLIC      G           EXECUTE     Y           SQLCAMESSAGE       SYSIBM        PROCEDURE
PUBLIC      G           EXECUTE     Y           SQLCAMESSAGECCSID  SYSIBM        PROCEDURE
```

Note: The SYSIBMADM.PRIVILEGES administrative view is available starting with version 9.1 of the DB2 database manager.

Procedure for releases earlier than version 9.1:

For releases earlier than version 9.1 of the DB2 database manager, during database creation, SELECT privilege on the system catalog views is granted to PUBLIC. In most cases, this does not present any security problems. For very sensitive data, however, it may be inappropriate, as these tables describe every object in the database. If this is the case, consider revoking the SELECT privilege from PUBLIC; then grant the SELECT privilege as required to specific users. Granting and revoking SELECT on the system catalog views is done in the same way as for any view, but you must have either SYSADM or DBADM authority to do this.

At a minimum, if you don't want any user to be able to know what objects other users have access to, you should consider restricting access to the following catalog and administrative views:
- SYSCAT.COLAUTH
- SYSCAT.DBAUTH
- SYSCAT.INDEXAUTH
- SYSCAT.PACKAGEAUTH
- SYSCAT.PASSTHRUAUTH
- SYSCAT.ROUTINEAUTH
- SYSCAT.SCHEMAAUTH
- SYSCAT.SECURITYLABELACCESS
- SYSCAT.SECURITYPOLICYEXEMPTIONS
- SYSCAT.SEQUENCEAUTH
- SYSCAT.SURROGATEAUTHIDS
- SYSCAT.TABAUTH
- SYSCAT.TBSPACEAUTH
- SYSCAT.XSROBJECTAUTH
- SYSIBMADM.AUTHORIZATIONIDS
- SYSIBMADM.OBJECTOWNERS
- SYSIBMADM.PRIVILEGES

This would prevent information on user privileges from becoming available to everyone with access to the database.

You should also examine the columns for which statistics are gathered. Some of the statistics recorded in the system catalog contain data values which could be sensitive information in your environment. If these statistics contain sensitive data, you may wish to revoke SELECT privilege from PUBLIC for the SYSCAT.COLUMNS and SYSCAT.COLDIST catalog views.

If you wish to limit access to the system catalog views, you could define views to let each authorization name retrieve information about its own privileges.

For example, the following view MYSELECTS includes the owner and name of every table on which a user's authorization name has been directly granted SELECT privilege:

```
CREATE VIEW MYSELECTS AS
   SELECT TABSCHEMA, TABNAME FROM SYSCAT.TABAUTH
   WHERE GRANTEETYPE = 'U'
     AND GRANTEE = USER
     AND SELECTAUTH = 'Y'
```

The keyword USER in this statement is equal to the value of the current session authorization name.

The following statement makes the view available to every authorization name:

```
GRANT SELECT ON TABLE MYSELECTS TO PUBLIC
```

And finally, remember to revoke SELECT privilege on the view and base table by issuing the following two statements:

```
REVOKE SELECT ON TABLE SYSCAT.TABAUTH FROM PUBLIC
REVOKE SELECT ON TABLE SYSIBM.SYSTABAUTH FROM PUBLIC
```

Related concepts:
- "Catalog statistics" in *Performance Guide*
- "Database authorities" on page 74
- "Using the system catalog for security issues" on page 161

Related tasks:
- "Granting privileges" on page 95
- "Revoking privileges" on page 97

Related reference:
- "CREATE DATABASE " on page 461
- "PRIVILEGES administrative view ÔÇô Retrieve privilege information" in *Administrative SQL Routines and Views*

System catalog views

System catalog views

The database manager creates and maintains two sets of system catalog views that are defined on top of the base system catalog tables.
- SYSCAT views are read-only catalog views that are found in the SYSCAT schema. SELECT privilege on these views is granted to PUBLIC by default.
- SYSSTAT views are updatable catalog views that are found in the SYSSTAT schema. The updatable views contain statistical information that is used by the optimizer. The values in some columns in these views can be changed to test performance. (Before changing any statistics, it is recommended that the RUNSTATS command be invoked so that all the statistics reflect the current state.)

Applications should be written to the SYSCAT and SYSSTAT views rather than the base catalog tables.

All the system catalog views are created at database creation time. The catalog views cannot be explicitly created or dropped. The views are updated during normal operation in response to SQL data definition statements, environment routines, and certain utilities. Data in the system catalog views is available through normal SQL query facilities. The system catalog views (with the exception of some updatable catalog views) cannot be modified using normal SQL data manipulation statements.

An object (table, column, function, or index) will appear in a user's updatable catalog view only if that user created the object, holds CONTROL privilege on the object, or holds explicit DBADM authority.

The order of columns in the views may change from release to release. To prevent this from affecting programming logic, specify the columns in a select list explicitly, and avoid using `SELECT *`. Columns have consistent names based on the types of objects that they describe.

Described Object	**Column Names**
Table	TABSCHEMA, TABNAME
Index	INDSCHEMA, INDNAME
View	VIEWSCHEMA, VIEWNAME
Constraint	CONSTSCHEMA, CONSTNAME
Trigger	TRIGSCHEMA, TRIGNAME
Package	PKGSCHEMA, PKGNAME
Type	TYPESCHEMA, TYPENAME, TYPEID
Function	ROUTINESCHEMA, ROUTINENAME, ROUTINEID
Method	ROUTINESCHEMA, ROUTINENAME, ROUTINEID
Procedure	ROUTINESCHEMA, ROUTINENAME, ROUTINEID
Column	COLNAME
Schema	SCHEMANAME
Table Space	TBSPACE
Database partition group	NGNAME
Buffer pool	BPNAME
Event Monitor	EVMONNAME
Creation Timestamp	CREATE_TIME

Road map to the catalog views

Table 31. Road map to the read-only catalog views

Description	Catalog View
attributes of structured data types	
authorities on database	"SYSCAT.DBAUTH " on page 173
buffer pool configuration on database partition group	

Table 31. Road map to the read-only catalog views (continued)

Description	Catalog View
buffer pool size on database partition	
cast functions	
check constraints	
column privileges	"SYSCAT.COLAUTH " on page 173
columns	
columns referenced by check constraints	
columns used in dimensions	
columns used in keys	
constraint dependencies	
database partition group database partitions	
database partition group definitions	
data partitions	
data types	
detailed column group statistics	
detailed column options	
detailed column statistics	
distribution maps	
event monitor definitions	
events currently monitored	
function dependencies[1]	
function mapping	
function mapping options	
function parameter mapping options	
function parameters[1]	
functions[1]	
hierarchies (types, tables, views)	
identity columns	
index columns	
index dependencies	
index exploitation	
index extension dependencies	
index extension parameters	
index extension search methods	
index extensions	

Table 31. Road map to the read-only catalog views (continued)

Description	Catalog View
index options	
index privileges	“SYSCAT.INDEXAUTH ” on page 174
indexes	
method dependencies[1]	
method parameters[1]	
methods[1]	
nicknames	
object mapping	
package dependencies	“SYSCAT.PACKAGEDEP ” on page 175
package privileges	“SYSCAT.PACKAGEAUTH ” on page 175
packages	
partitioned tables	
pass-through privileges	“SYSCAT.PASSTHRUAUTH ” on page 176
predicate specifications	
procedure options	
procedure parameter options	
procedure parameters[1]	
procedures[1]	
protected tables	“SYSCAT.SECURITYLABELACCESS ” on page 179
	“SYSCAT.SECURITYLABELCOMPONENTELEMENTS ” on page 180
	“SYSCAT.SECURITYLABELCOMPONENTS ” on page 180
	“SYSCAT.SECURITYLABELS ” on page 180
	“SYSCAT.SECURITYPOLICIES ” on page 181
	“SYSCAT.SECURITYPOLICYCOMPONENTRULES ” on page 181
	“SYSCAT.SECURITYPOLICYEXEMPTIONS ” on page 182
	“SYSCAT.SURROGATEAUTHIDS ” on page 182
provides DB2 Universal Database for z/OS and OS/390 compatibility	
referential constraints	
remote table options	
routine dependencies	
routine parameters[1]	
routine privileges	“SYSCAT.ROUTINEAUTH ” on page 177
routines[1]	
schema privileges	“SYSCAT.SCHEMAAUTH ” on page 177
schemas	“SYSCAT.SCHEMATA ” on page 178
sequence privileges	“SYSCAT.SEQUENCEAUTH ” on page 179

Table 31. Road map to the read-only catalog views (continued)

Description	Catalog View
sequences	"SYSCAT.SEQUENCES " on page 183
server options	
server options values	"SYSCAT.USEROPTIONS " on page 192
statements in packages	
stored procedures	
system servers	
table constraints	"SYSCAT.TABCONST " on page 184
table dependencies	
table privileges	"SYSCAT.TABAUTH " on page 193
table space use privileges	"SYSCAT.TBSPACEAUTH " on page 192
table spaces	"SYSCAT.TABLESPACES " on page 191
tables	"SYSCAT.TABLES " on page 185
transforms	
trigger dependencies	
triggers	
type mapping	
user-defined functions	
view dependencies	
views	"SYSCAT.TABLES " on page 185
wrapper options	
wrappers	
XML values index	
XSR objects	

[1] The catalog views for functions, methods, and procedures from DB2 Version 7.1 and earlier still exist. These views, however, do not reflect any changes since DB2 Version 7.1. The views are:

```
Functions:  SYSCAT.FUNCTIONS, SYSCAT.FUNCDEP, SYSCAT.FUNCPARMS
Methods:    SYSCAT.FUNCTIONS, SYSCAT.FUNCDEP, SYSCAT.FUNCPARMS
Procedures: SYSCAT.PROCEDURES, SYSCAT.PROCPARMS
```

Table 32. Road map to the updatable catalog views

Description	Catalog View
columns	

Table 32. Road map to the updatable catalog views (continued)

Description	Catalog View
detailed column group statistics	
detailed column statistics	
indexes	
routines[1]	
tables	

[1] The SYSSTAT.FUNCTIONS catalog view still exists for updating the statistics for functions and methods. This view, however, does not reflect any changes since DB2 Version 7.1.

SYSCAT.COLAUTH

Each row represents a user or a group that has been granted one or more privileges on a column.

Table 33. SYSCAT.COLAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of a privilege.
GRANTEE	VARCHAR(128)		Holder of a privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
TABSCHEMA	VARCHAR(128)		Schema name of the table or view on which the privilege is held.
TABNAME	VARCHAR(128)		Unqualified name of the table or view on which the privilege is held.
COLNAME	VARCHAR(128)		Name of the column to which this privilege applies.
COLNO	SMALLINT		Column number of this column within the table (starting with 0).

PRIVTYPE	CHAR(1)		R = Reference privilege
			U = Update privilege

GRANTABLE	CHAR(1)		G = Privilege is grantable N = Privilege is not grantable

Notes:

1. Privileges can be granted by column, but can be revoked only on a table-wide basis.

SYSCAT.DBAUTH

Each row represents a user or a group that has been granted one or more database-level authorities.

Table 34. SYSCAT.DBAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the authority.
GRANTEE	VARCHAR(128)		Holder of the authority.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user

BINDADDAUTH	CHAR(1)		Authority to create packages.
			N = Not held
			Y = Held

CONNECTAUTH	CHAR(1)		Authority to connect to the database.
			N = Not held
			Y = Held

CREATETABAUTH	CHAR(1)		Authority to create tables.
			N = Not held
			Y = Held

DBADMAUTH	CHAR(1)		DBADM authority.
			N = Not held
			Y = Held

EXTERNALROUTINEAUTH	CHAR(1)		Authority to create external routines.
			N = Not held
			Y = Held

IMPLSCHEMAAUTH	CHAR(1)		Authority to implicitly create schemas by creating objects in non-existent schemas.
			N = Not held
			Y = Held

LOADAUTH	CHAR(1)		Authority to use the DB2 load utility.
			N = Not held
			Y = Held

NOFENCEAUTH	CHAR(1)		Authority to create non-fenced user-defined functions.
			N = Not held
			Y = Held

QUIESCECONNECTAUTH	CHAR(1)		Authority to access the database when it is quiesced.
			N = Not held
			Y = Held

LIBRARYADMAUTH	CHAR(1)		Reserved for future use.

SECURITYADMAUTH	CHAR(1)		Security Administrator authority.
			N = Not held
			Y = Held

SYSCAT.INDEXAUTH

Each row represents a user or group that has been granted CONTROL privilege on an index.

Table 35. SYSCAT.INDEXAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the privilege.
GRANTEE	VARCHAR(128)		Holder of the privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
INDSCHEMA	VARCHAR(128)		Schema name of the index.
INDNAME	VARCHAR(128)		Unqualified name of the index.

CONTROLAUTH	CHAR(1)		CONTROL privilege.
			N = Not held
			Y = Held

SYSCAT.PACKAGEAUTH

Each row represents a user or group that has been granted one or more privileges on a package.

Table 36. SYSCAT.PACKAGEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the privilege.
GRANTEE	VARCHAR(128)		Holder of the privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
PKGSCHEMA	VARCHAR(128)		Schema name of the package.
PKGNAME	VARCHAR(128)		Unqualified name of the package.

CONTROLAUTH	CHAR(1)		CONTROL privilege.
			N = Not held
			Y = Held

BINDAUTH	CHAR(1)		Privilege to bind the package.
			G = Held and grantable
			N = Not held
			Y = Held

EXECUTEAUTH	CHAR(1)		Privilege to execute the package.
			G = Held and grantable
			N = Not held
			Y = Held

SYSCAT.PACKAGEDEP

Each row represents a dependency of a package on some other object. The package depends on the object of type BTYPE of name BNAME, so a change to the object affects the package.

Table 37. SYSCAT.PACKAGEDEP Catalog View

Column Name	Data Type	Nullable	Description
PKGSCHEMA	VARCHAR(128)		Schema name of the package.

Table 37. SYSCAT.PACKAGEDEP Catalog View (continued)

Column Name	Data Type	Nullable	Description
PKGNAME	VARCHAR(128)		Unqualified name of the package.
BINDER	VARCHAR(128)		Binder of the package.

BTYPE	CHAR(1)		Type of object on which there is a dependency. Possible values are:
			A = Alias
			B = Trigger
			D = Server definition
			F = Routine instance
			I = Index
			M = Function mapping
			N = Nickname
			O = Privilege dependency on all subtables or subviews in a table or view hierarchy
			P = Page size
			Q = Sequence object
			R = Structured type
			S = Materialized query table
			T = Table (untyped)
			U = Typed table
			V = View (untyped)
			W = Typed view
			Z = XSR object

BSCHEMA	VARCHAR(128)		Schema name of an object on which the package depends.
BNAME	VARCHAR(128)		Unqualified name of an object on which the package depends.
TABAUTH	SMALLINT	Y	If BTYPE is ’O’, ’S’, ’T’, ’U’, ’V’, or ’W’, encodes the privileges that are required by this package (SELECT, INSERT, UPDATE, or DELETE).
UNIQUE_ID	CHAR(8) FOR BIT DATA		Identifier for a specific package when multiple packages having the same name exist.
PKGVERSION	VARCHAR(64)		Version identifier for the package.

Notes:

1. If a function instance with dependencies is dropped, the package is put into an ″inoperative″ state, and it must be explicitly rebound. If any other object with dependencies is dropped, the package is put into an ″invalid″ state, and the system will attempt to rebind the package automatically when it is first referenced.

SYSCAT.PASSTHRUAUTH

Each row represents a user or group that has been granted pass-through authorization to query a data source.

Table 38. SYSCAT.PASSTHRUAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the privilege.
GRANTEE	VARCHAR(128)		Holder of the privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
SERVERNAME	VARCHAR(128)		Name of the data source to which authorization is being granted.

SYSCAT.SCHEMAAUTH

Each row represents a user or group that has been granted one or more privileges on a schema.

Table 39. SYSCAT.SCHEMAAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of a privilege.
GRANTEE	VARCHAR(128)		Holder of a privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
SCHEMANAME	VARCHAR(128)		Name of the schema to which this privilege applies.

ALTERINAUTH	CHAR(1)		Privilege to alter or comment on objects in the named schema.
			G = Held and grantable
			N = Not held
			Y = Held

CREATEINAUTH	CHAR(1)		Privilege to create objects in the named schema.
			G = Held and grantable
			N = Not held
			Y = Held

DROPINAUTH	CHAR(1)		Privilege to drop objects from the named schema.
			G = Held and grantable
			N = Not held
			Y = Held

SYSCAT.ROUTINEAUTH

Each row represents a user or group that has been granted EXECUTE privilege on a particular routine (function, method, or procedure), or on all routines in a particular schema in the database.

Table 40. SYSCAT.ROUTINEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the privilege. 'SYSIBM' if the privilege was granted by the system.
GRANTEE	VARCHAR(128)		Holder of the privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
SCHEMA	VARCHAR(128)		Schema name of the routine.
SPECIFICNAME	VARCHAR(128)	Y	Specific name of the routine. If SPECIFICNAME is null and ROUTINETYPE is not 'M', the privilege applies to all routines of the type specified in ROUTINETYPE in the schema specified in SCHEMA. If SPECIFICNAME is null and ROUTINETYPE is 'M', the privilege applies to all methods for the subject type specified by TYPENAME in the schema specified by TYPESCHEMA. If SPECIFICNAME is null, ROUTINETYPE is 'M', and both TYPENAME and TYPESCHEMA are null, the privilege applies to all methods for all types in the schema.
TYPESCHEMA	VARCHAR(128)	Y	Schema name of the type for the method. Null if ROUTINETYPE is not 'M'.
TYPENAME	VARCHAR(128)	Y	Unqualified name of the type for the method. Null if ROUTINETYPE is not 'M'. If TYPENAME is null and ROUTINETYPE is 'M', the privilege applies to all methods for any subject type if they are in the schema specified by SCHEMA.

ROUTINETYPE	CHAR(1)		Type of the routine.
			F = Function
			M = Method
			P = Procedure

EXECUTEAUTH	CHAR(1)		Privilege to execute the routine.
			G = Held and grantable
			N = Not held
			Y = Held

GRANT_TIME	TIMESTAMP		Time at which the privilege was granted.

SYSCAT.SCHEMATA

Each row represents a schema.

Table 41. SYSCAT.SCHEMATA Catalog View

Column Name	Data Type	Nullable	Description
SCHEMANAME	VARCHAR(128)		Name of the schema.
OWNER	VARCHAR(128)		Authorization ID of the schema, who has the authority to drop the schema and all objects within it. The value for implicitly created schemas is 'SYSIBM'.

Table 41. SYSCAT.SCHEMATA Catalog View (continued)

Column Name	Data Type	Nullable	Description
DEFINER	VARCHAR(128)		Authorization ID under which the schema was created.
CREATE_TIME	TIMESTAMP		Time at which the schema was created.
REMARKS	VARCHAR(254)	Y	User-provided comments, or null.

SYSCAT.SEQUENCEAUTH

Each row represents a user or group that has been granted one or more privileges on a sequence.

Table 42. SYSCAT.SEQUENCEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of a privilege.
GRANTEE	VARCHAR(128)		Holder of a privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
SEQSCHEMA	VARCHAR(128)		Schema name of the sequence.
SEQNAME	VARCHAR(128)		Unqualified name of the sequence.

ALTERAUTH	CHAR(1)		Privilege to alter the sequence.
			G = Held and grantable
			N = Not held
			Y = Held

USAGEAUTH	CHAR(1)		Privilege to reference the sequence.
			G = Held and grantable
			N = Not held
			Y = Held

SYSCAT.SECURITYLABELACCESS

Each row represents a security label that was granted to the database authorization ID.

Table 43. SYSCAT.SECURITYLABELACCESS Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the security label.
GRANTEE	VARCHAR(128)		Holder of the security label.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user
SECLABELID	INTEGER		Identifier for the security label.
SECPOLICYID	INTEGER		Identifier for the security policy that is associated with the security label.

ACCESSTYPE	CHAR(1)		B = Both read and write access
			R = Read access
			W = Write access

Table 43. SYSCAT.SECURITYLABELACCESS Catalog View (continued)

Column Name	Data Type	Nullable	Description
GRANT_TIME	TIMESTAMP		Time at which the security label was granted.

SYSCAT.SECURITYLABELCOMPONENTELEMENTS

Each row represents an element value for a security label component.

Table 44. SYSCAT.SECURITYLABELCOMPONENTELEMENTS Catalog View

Column Name	Data Type	Nullable	Description
COMPID	INTEGER		Identifier for the security label component.
ELEMENTVALUE	VARCHAR(32)		Element value for the security label component.
ELEMENTVALUEENCODING	CHAR(8) FOR BIT DATA		Encoded form of the element value.
PARENTELEMENTVALUE	VARCHAR(32)	Y	Name of the parent of an element for tree components; null for set and array components, and for the ROOT node of a tree component.

SYSCAT.SECURITYLABELCOMPONENTS

Each row represents a security label component.

Table 45. SYSCAT.SECURITYLABELCOMPONENTS Catalog View

Column Name	Data Type	Nullable	Description
COMPNAME	VARCHAR(128)		Name of the security label component.
COMPID	INTEGER		Identifier for the security label component.

COMPTYPE	CHAR(1)		Security label component type.
			A = Array
			S = Set
			T = Tree

NUMELEMENTS	INTEGER		Number of elements in the security label component.
CREATE_TIME	TIMESTAMP		Time at which the security label component was created.
REMARKS	VARCHAR(254)		User-provided comments, or null.

SYSCAT.SECURITYLABELS

Each row represents a security label.

Table 46. SYSCAT.SECURITYLABELS Catalog View

Column Name	Data Type	Nullable	Description
SECLABELNAME	VARCHAR(128)		Name of the security label.
SECLABELID	INTEGER		Identifier for the security label.

Table 46. SYSCAT.SECURITYLABELS Catalog View (continued)

Column Name	Data Type	Nullable	Description
SECPOLICYID	INTEGER		Identifier for the security policy to which the security label belongs.
SECLABEL	SYSPROC.DB2SECURITYLABEL		Internal representation of the security label.
CREATE_TIME	TIMESTAMP		Time at which the security label was created.
REMARKS	VARCHAR(254)		User-provided comments, or null.

SYSCAT.SECURITYPOLICIES

Each row represents a security policy.

Table 47. SYSCAT.SECURITYPOLICIES Catalog View

Column Name	Data Type	Nullable	Description
SECPOLICYNAME	VARCHAR(128)		Name of the security policy.
SECPOLICYID	INTEGER		Identifier for the security policy.
NUMSECLABELCOMP	INTEGER		Number of security label components in the security policy.
RWSECLABELREL	CHAR(1)		Relationship between the security labels for read and write access granted to the same authorization ID. S = The security label for write access granted to a user is a subset of the security label for read access granted to that same user

NOTAUTHWRITESECLABEL	CHAR(1)		Action to take when a user is not authorized to write the security label that is specified in the INSERT or UPDATE statement.
			O = Override
			R = Restrict

CREATE_TIME	TIMESTAMP		Time at which the security policy was created.
REMARKS	VARCHAR(254)		User-provided comments, or null.

SYSCAT.SECURITYPOLICYCOMPONENTRULES

Each row represents the read and write access rules for a security label component of the security policy.

Table 48. SYSCAT.SECURITYPOLICYCOMPONENTRULES Catalog View

Column Name	Data Type	Nullable	Description
SECPOLICYID	INTEGER		Identifier for the security policy.
COMPID	INTEGER		Identifier for the security label component of the security policy.
ORDINAL	INTEGER		Position of the security label component as it appears in the security policy, starting with 1.
READACCESSRULENAME	VARCHAR(128)		Name of the read access rule that is associated with the security label component.

Table 48. SYSCAT.SECURITYPOLICYCOMPONENTRULES Catalog View (continued)

Column Name	Data Type	Nullable	Description
READACCESSRULETEXT	VARCHAR(512)		Text of the read access rule that is associated with the security label component.
WRITEACCESSRULENAME	VARCHAR(128)		Name of the write access rule that is associated with the security label component.
WRITEACCESSRULETEXT	VARCHAR(512)		Text of the write access rule that is associated with the security label component.

SYSCAT.SECURITYPOLICYEXEMPTIONS

Each row represents a security policy exemption that was granted to a database authorization ID.

Table 49. SYSCAT.SECURITYPOLICYEXEMPTIONS Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the exemption.
GRANTEE	VARCHAR(128)		Holder of the exemption.
GRANTEETYPE	CHAR(1)		U = Grantee is an individual user
SECPOLICYID	INTEGER		Identifier for the security policy for which the exemption was granted.
ACCESSRULENAME	VARCHAR(128)		Name of the access rule for which the exemption was granted.

ACCESSTYPE	CHAR(1)		Type of access to which the rule applies.
			R = Read access
			W = Write access

ORDINAL	INTEGER		Position of the security label component in the security policy to which the rule applies.
ACTIONALLOWED	CHAR(1)		If the rule is DB2LBACWRITEARRAY, then: B = Write down and write up D = Write down U = Write up Blank otherwise.
GRANT_TIME	TIMESTAMP		Time at which the exemption was granted.

SYSCAT.SURROGATEAUTHIDS

Each row represents an authorization ID for which another authorization ID can act as a surrogate.

Table 50. SYSCAT.SURROGATEAUTHIDS Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Authorization ID that granted TRUSTEDID the ability to act as a surrogate.
TRUSTEDID	VARCHAR(128)		Identifier for the entity that is trusted to act as a surrogate.

Table 50. SYSCAT.SURROGATEAUTHIDS Catalog View (continued)

Column Name	Data Type	Nullable	Description

TRUSTEDIDTYPE	CHAR(1)		G = Group
			U = User

SURROGATEAUTHID	VARCHAR(128)		Surrogate authorization ID that can be assumed by TRUSTEDID. 'PUBLIC' indicates that TRUSTEDID can assume any authorization ID.

SURROGATEAUTHIDTYPE	CHAR(1)		G = Group
			U = User

GRANT_TIME	TIMESTAMP		Time at which the grant was made.

SYSCAT.SEQUENCES

Each row represents a sequence.

Table 51. SYSCAT.SEQUENCES Catalog View

Column Name	Data Type	Nullable	Description
SEQSCHEMA	VARCHAR(128)		Schema name of the sequence.
SEQNAME	VARCHAR(128)		Unqualified name of the sequence.
DEFINER[1]	VARCHAR(128)		Authorization ID under which the sequence was created.
OWNER	VARCHAR(128)		Authorization ID under which the sequence was created.
SEQID	INTEGER		Identifier for the sequence.

SEQTYPE	CHAR(1)		Type of sequence.
			I = Identity sequence
			S = Regular sequence

INCREMENT	DECIMAL(31,0)		Increment value.
START	DECIMAL(31,0)		Start value of the sequence.
MAXVALUE	DECIMAL(31,0)		Maximum value of the sequence.
MINVALUE	DECIMAL(31,0)		Minimum value of the sequence.
NEXTCACHEFIRSTVALUE	DECIMAL(31,0)	Y	The first value available to be assigned in the next cache block. If no caching, the next value available to be assigned.

CYCLE	CHAR(1)		Indicates whether or not the sequence can continue to generate values after reaching its maximum or minimum value.
			N = Sequence cannot cycle
			Y = Sequence can cycle

CACHE	INTEGER		Number of sequence values to pre-allocate in memory for faster access. 0 indicates that values of the sequence are not to be preallocated. In a partitioned database, this value applies to each database partition.

Table 51. SYSCAT.SEQUENCES Catalog View (continued)

Column Name	Data Type	Nullable	Description
ORDER	CHAR(1)		Indicates whether or not the sequence numbers must be generated in order of request. N = Sequence numbers are not required to be generated in order of request Y = Sequence numbers must be generated in order of request
DATATYPEID	INTEGER		For built-in types, the internal identifier of the built-in type. For distinct types, the internal identifier of the distinct type.
SOURCETYPEID	INTEGER		For a built-in type, this has a value of 0. For a distinct type, this is the internal identifier of the built-in type that is the source type for the distinct type.
CREATE_TIME	TIMESTAMP		Time at which the sequence was created.
ALTER_TIME	TIMESTAMP		Time at which the sequence was last altered.
PRECISION	SMALLINT		Precision of the data type of the sequence. Possible values are: 5 = SMALLINT 10 = INTEGER 19 = BIGINT For DECIMAL, it is the precision of the specified DECIMAL data type.

ORIGIN	CHAR(1)		Origin of the sequence.
			S = System-generated sequence
			U = User-generated sequence

REMARKS	VARCHAR(254)		User-provided comments, or null.

Notes:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABCONST

Each row represents a table constraint of type CHECK, UNIQUE, PRIMARY KEY, or FOREIGN KEY. For table hierarchies, each constraint is recorded only at the level of the hierarchy where the constraint was created.

Table 52. SYSCAT.TABCONST Catalog View

Column Name	Data Type	Nullable	Description
CONSTNAME	VARCHAR(128)		Name of the constraint.
TABSCHEMA	VARCHAR(128)		Schema name of the table to which this constraint applies.
TABNAME	VARCHAR(128)		Unqualified name of the table to which this constraint applies.
OWNER	VARCHAR(128)		Authorization ID under which the constraint was created.

Table 52. SYSCAT.TABCONST Catalog View (continued)

Column Name	Data Type	Nullable	Description

TYPE	CHAR(1)		Indicates the constraint type.
			F = Foreign key
			I = Functional dependency
			K = Check
			P = Primary key
			U = Unique

ENFORCED	CHAR(1)		N = Do not enforce constraint Y = Enforce constraint
CHECKEXISTINGDATA	CHAR(1)		D = Defer checking any existing data I = Immediately check existing data N = Never check existing data
ENABLEQUERYOPT	CHAR(1)		N = Query optimization is disabled Y = Query optimization is enabled
DEFINER[1]	VARCHAR(128)		Authorization ID under which the constraint was created.
REMARKS	VARCHAR(254)	Y	User-provided comments, or null.

Notes:
1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABLES

Each row represents a table, view, alias, or nickname. Each table or view hierarchy has one additional row representing the hierarchy table or hierarchy view that implements the hierarchy. Catalog tables and views are included.

Table 53. SYSCAT.TABLES Catalog View

Column Name	Data Type	Nullable	Description
TABSCHEMA	VARCHAR(128)		Schema name of the object.
TABNAME	VARCHAR(128)		Unqualified name of the object.
OWNER	VARCHAR(128)		Authorization ID under which the table, view, alias, or nickname was created.

TYPE	CHAR(1)		Type of object.
			A = Alias
			G = Global temporary table
			H = Hierarchy table
			L = Detached table
			N = Nickname
			S = Materialized query table
			T = Table (untyped)
			U = Typed table
			V = View (untyped)
			W = Typed view

Table 53. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description

STATUS	CHAR(1)		Status of the object.
			C = Set integrity pending
			N = Normal
			X = Inoperative

BASE_TABSCHEMA	VARCHAR(128)	Y	If TYPE = 'A', contains the schema name of the table, view, alias, or nickname that is referenced by this alias; null value otherwise.
BASE_TABNAME	VARCHAR(128)	Y	If TYPE = 'A', contains the unqualified name of the table, view, alias, or nickname that is referenced by this alias; null value otherwise.
ROWTYPESCHEMA	VARCHAR(128)	Y	Schema name of the row type for this table, if applicable; null value otherwise.
ROWTYPENAME	VARCHAR(128)	Y	Unqualified name of the row type for this table, if applicable; null value otherwise.
CREATE_TIME	TIMESTAMP		Time at which the object was created.
INVALIDATE_TIME	TIMESTAMP		Time at which the object was last invalidated.
STATS_TIME	TIMESTAMP	Y	Time at which any change was last made to recorded statistics for this object. Null if statistics are not collected.
COLCOUNT	SMALLINT		Number of columns, including inherited columns (if any).
TABLEID	SMALLINT		Internal logical object identifier.
TBSPACEID	SMALLINT		Internal logical identifier for the primary table space for this object.
CARD	BIGINT		Total number of rows; -1 if statistics are not collected.
NPAGES	BIGINT		Total number of pages on which the rows of the table exist; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
FPAGES	BIGINT		Total number of pages; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
OVERFLOW	BIGINT		Total number of overflow records in the table; -1 for a view or alias, or if statistics are not collected; -2 for a subtable or hierarchy table.
TBSPACE	VARCHAR(128)	Y	Name of the primary table space for the table. If no other table space is specified, all parts of the table are stored in this table space. Null for aliases, views, and partitioned tables.
INDEX_TBSPACE	VARCHAR(128)	Y	Name of the table space that holds all indexes created on this table. Null for aliases, views, and partitioned tables, or if the INDEX IN clause was omitted or specified with the same value as the IN clause of the CREATE TABLE statement.

Table 53. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
LONG_TBSPACE	VARCHAR(128)	Y	Name of the table space that holds all long data (LONG or LOB column types) for this table. Null for aliases, views, and partitioned tables, or if the LONG IN clause was omitted or specified with the same value as the IN clause of the CREATE TABLE statement.
PARENTS	SMALLINT	Y	Number of parent tables for this object; that is, the number of referential constraints in which this object is a dependent.
CHILDREN	SMALLINT	Y	Number of dependent tables for this object; that is, the number of referential constraints in which this object is a parent.
SELFREFS	SMALLINT	Y	Number of self-referencing referential constraints for this object; that is, the number of referential constraints in which this object is both a parent and a dependent.
KEYCOLUMNS	SMALLINT	Y	Number of columns in the primary key.
KEYINDEXID	SMALLINT	Y	Index identifier for the primary key index; 0 or the null value if there is no primary key.
KEYUNIQUE	SMALLINT		Number of unique key constraints (other than the primary key constraint) defined on this object.
CHECKCOUNT	SMALLINT		Number of check constraints defined on this object.
DATACAPTURE	CHAR(1)		L = Table participates in data replication, including replication of LONG VARCHAR and LONG VARGRAPHIC columns N = Table does not participate in data replication Y = Table participates in data replication, excluding replication of LONG VARCHAR and LONG VARGRAPHIC columns

Table 53. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
CONST_CHECKED	CHAR(32)		Byte 1 represents foreign key constraint. Byte 2 represents check constraint. Byte 5 represents materialized query table. Byte 6 represents generated column. Byte 7 represents staging table. Byte 8 represents data partitioning constraint. Other bytes are reserved for future use. Possible values are: F = In byte 5, the materialized query table cannot be refreshed incrementally. In byte 7, the content of the staging table is incomplete and cannot be used for incremental refresh of the associated materialized query table. N = Not checked U = Checked by user W = Was in 'U' state when the table was placed in set integrity pending state Y = Checked by system
PMAP_ID	SMALLINT	Y	Identifier for the distribution map that is currently in use by this table (null for aliases or views).

PARTITION_MODE	CHAR(1)		Indicates how data is distributed among database partitions in a partitioned database system.
			H = Hashing
			R = Replicated across database partitions
			Blank = No database partitioning

LOG_ATTRIBUTE	CHAR(1)		Always 0. This column is no longer used.
PCTFREE	SMALLINT		Percentage of each page to be reserved for future inserts.
APPEND_MODE	CHAR(1)		Controls how rows are inserted into pages. N = New rows are inserted into existing spaces, if available Y = New rows are appended to the end of the data Initial value is 'N'.

REFRESH	CHAR(1)		Refresh mode.
			D = Deferred
			I = Immediate
			O = Once
			Blank = Not a materialized query table

REFRESH_TIME	TIMESTAMP	Y	For REFRESH = 'D' or 'O', time at which the data was last refreshed (REFRESH TABLE statement); null value otherwise.

Table 53. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
LOCKSIZE	CHAR(1)		Indicates the preferred lock granularity for tables that are accessed by data manipulation language (DML) statements. Applies to tables only. Possible values are: I = Block insert; R = Row; T = Table; Blank = Not applicable. Initial value is 'R'.
VOLATILE	CHAR(1)		C = Cardinality of the table is volatile; Blank = Not applicable
ROW_FORMAT	CHAR(1)		Not used.
PROPERTY	VARCHAR(32)		Properties for a table. A single blank indicates that the table has no properties.
STATISTICS_PROFILE	CLOB(10M)	Y	RUNSTATS command used to register a statistical profile for the object.
COMPRESSION	CHAR(1)		B = Both value and row compression are activated; N = No compression is activated; a row format that does not support compression is used; R = Row compression is activated; a row format that supports compression might be used; V = Value compression is activated; a row format that supports compression is used; Blank = Not applicable
ACCESS_MODE	CHAR(1)		Access restriction state of the object. These states only apply to objects that are in set integrity pending state or to objects that were processed by a SET INTEGRITY statement. Possible values are: D = No data movement; F = Full access; N = No access; R = Read-only access
CLUSTERED	CHAR(1)	Y	Y = Table is multidimensionally clustered (even if only by one dimension); Null value = Table is not multidimensionally clustered
ACTIVE_BLOCKS	BIGINT		Total number of active blocks in the table, or -1. Applies to multidimensional clustering (MDC) tables only.
DROPRULE	CHAR(1)		N = No rule; R = Restrict rule applies on drop
MAXFREESPACESEARCH	SMALLINT		Reserved for future use.

Table 53. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
AVGCOMPRESSEDROWSIZE	SMALLINT		Average length (in bytes) of compressed rows in this table; -1 if statistics are not collected.
AVGROWCOMPRESSIONRATIO	REAL		For compressed rows in the table, this is the average compression ratio by row; that is, the average uncompressed row length divided by the average compressed row length; -1 if statistics are not collected.
AVGROWSIZE	SMALLINT		Average length (in bytes) of both compressed and uncompressed rows in this table; -1 if statistics are not collected.
PCTROWSCOMPRESSED	REAL		Compressed rows as a percentage of the total number of rows in the table; -1 if statistics are not collected.
LOGINDEXBUILD	VARCHAR(3)	Y	Level of logging that is to be performed during create, recreate, or reorganize index operations on the table. OFF = Index build operations on the table will be logged minimally ON = Index build operations on the table will be logged completely Null value = Value of the *logindexbuild* database configuration parameter will be used to determine whether or not index build operations are to be completely logged
CODEPAGE	SMALLINT		Code page of the object. This is the default code page used for all character columns, triggers, check constraints, and expression-generated columns.

ENCODING_SCHEME	CHAR(1)		A = CCSID ASCII was specified
			U = CCSID UNICODE was specified
			Blank = CCSID clause was not specified

PCTPAGESSAVED	SMALLINT		Approximate percentage of pages saved in the table as a result of row compression. This value includes overhead bytes for each user data row in the table, but does not include the space that is consumed by dictionary overhead; -1 if statistics are not collected.
LAST_REGEN_TIME	TIMESTAMP		Time at which any views or check constraints on the table were last regenerated.
SECPOLICYID	INTEGER		Identifier for the security policy protecting the table; 0 for non-protected tables.

PROTECTIONGRANULARITY	CHAR(1)		B = Both column- and row-level granularity
			C = Column-level granularity
			R = Row-level granularity
			Blank = Non-protected table

Table 53. SYSCAT.TABLES Catalog View (continued)

Column Name	Data Type	Nullable	Description
DEFINER[1]	VARCHAR(128)		Authorization ID under which the table, view, alias, or nickname was created.
REMARKS	VARCHAR(254)	Y	User-provided comments, or null.

Notes:

1. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TABLESPACES

Each row represents a table space.

Table 54. SYSCAT.TABLESPACES Catalog View

Column Name	Data Type	Nullable	Description
TBSPACE	VARCHAR(128)		Name of the table space.
OWNER	VARCHAR(128)		Authorization ID under which the table space was created.
CREATE_TIME	TIMESTAMP		Time at which the table space was created.
TBSPACEID	INTEGER		Identifier for the table space.

TBSPACETYPE	CHAR(1)		Type of table space.
			D = Database-managed space
			S = System-managed space

DATATYPE	CHAR(1)		Type of data that can be stored in this table space. A = All types of permanent data; regular table space L = All types of permanent data; large table space T = System temporary tables only U = Declared temporary tables only
EXTENTSIZE	INTEGER		Size of each extent, in pages of size PAGESIZE. This many pages are written to one container in the table space before switching to the next container.
PREFETCHSIZE	INTEGER		Number of pages of size PAGESIZE to be read when prefetching is performed; -1 when AUTOMATIC.
OVERHEAD	DOUBLE		Controller overhead and disk seek and latency time, in milliseconds (average for the containers in this table space).
TRANSFERRATE	DOUBLE		Time to read one page of size PAGESIZE into the buffer (average for the containers in this table space).
PAGESIZE	INTEGER		Size (in bytes) of pages in this table space.
DBPGNAME	VARCHAR(128)		Name of the database partition group that is associated with this table space.
BUFFERPOOLID	INTEGER		Identifier for the buffer pool that is used by this table space (1 indicates the default buffer pool).

Table 54. SYSCAT.TABLESPACES Catalog View (continued)

Column Name	Data Type	Nullable	Description
DROP_RECOVERY	CHAR(1)		Indicates whether or not tables in this table space can be recovered after a drop table operation. N = Tables are not recoverable Y = Tables are recoverable
NGNAME[1]	VARCHAR(128)		Name of the database partition group that is associated with this table space.
DEFINER[2]	VARCHAR(128)		Authorization ID under which the table space was created.
REMARKS	VARCHAR(254)	Y	User-provided comments, or null.

Notes:

1. The NGNAME column is included for backwards compatibility. See DBPGNAME.
2. The DEFINER column is included for backwards compatibility. See OWNER.

SYSCAT.TBSPACEAUTH

Each row represents a user or group that has been granted the USE privilege on a particular table space in the database.

Table 55. SYSCAT.TBSPACEAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the privilege.
GRANTEE	VARCHAR(128)		Holder of the privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
TBSPACE	VARCHAR(128)		Name of the table space.

USEAUTH	CHAR(1)		Privilege to create tables within the table space.
			G = Held and grantable
			N = Not held
			Y = Held

SYSCAT.USEROPTIONS

Each row represents a server-specific user option value.

Table 56. SYSCAT.USEROPTIONS Catalog View

Column Name	Data Type	Nullable	Description
AUTHID	VARCHAR(128)		Local authorization ID, in uppercase characters.
SERVERNAME	VARCHAR(128)		Name of the server on which the user is defined.
OPTION	VARCHAR(128)		Name of the user option.
SETTING	VARCHAR(2048)		Value of the user option.

SYSCAT.TABAUTH

Each row represents a user or group that has been granted one or more privileges on a table or view.

Table 57. SYSCAT.TABAUTH Catalog View

Column Name	Data Type	Nullable	Description
GRANTOR	VARCHAR(128)		Grantor of the privilege.
GRANTEE	VARCHAR(128)		Holder of the privilege.
GRANTEETYPE	CHAR(1)		G = Grantee is a group U = Grantee is an individual user
TABSCHEMA	VARCHAR(128)		Schema name of the table or view.
TABNAME	VARCHAR(128)		Unqualified name of the table or view.
CONTROLAUTH	CHAR(1)		CONTROL privilege. N = Not held Y = Held but not grantable

ALTERAUTH	CHAR(1)		Privilege to alter the table; allow a parent table to this table to drop its primary key or unique constraint; allow a table to become a materialized query table that references this table or view in the materialized query; or allow a table that references this table or view in its materialized query to no longer be a materialized query table.
			G = Held and grantable
			N = Not held
			Y = Held

DELETEAUTH	CHAR(1)		Privilege to delete rows from a table or updatable view.
			G = Held and grantable
			N = Not held
			Y = Held

INDEXAUTH	CHAR(1)		Privilege to create an index on a table.
			G = Held and grantable
			N = Not held
			Y = Held

INSERTAUTH	CHAR(1)		Privilege to insert rows into a table or updatable view, or to run the import utility against a table or view.
			G = Held and grantable
			N = Not held
			Y = Held

REFAUTH	CHAR(1)		Privilege to create and drop a foreign key referencing a table as the parent.
			G = Held and grantable
			N = Not held
			Y = Held

Table 57. SYSCAT.TABAUTH Catalog View (continued)

Column Name	Data Type	Nullable	Description

SELECTAUTH	CHAR(1)		Privilege to retrieve rows from a table or view, create views on a table, or to run the export utility against a table or view.
			G = Held and grantable
			N = Not held
			Y = Held

UPDATEAUTH	CHAR(1)		Privilege to run the UPDATE statement against a table or updatable view.
			G = Held and grantable
			N = Not held
			Y = Held

Part 5. Auditing database activities

Chapter 21. Auditing DB2 database activities

DB2 database auditing activities are shown in this section.

Introduction to the DB2 database audit facility

Authentication, authorities, and privileges can be used to control known or anticipated access to data, but these methods may be insufficient to prevent unknown or unanticipated access to data. To assist in the detection of this latter type of data access, DB2 database provides an audit facility. Successful monitoring of unwanted data access and subsequent analysis can lead to improvements in the control of data access and the ultimate prevention of malicious or careless unauthorized access to the data. The monitoring of application and individual user access, including system administration actions, can provide a historical record of activity on your database systems.

The DB2 database audit facility generates, and allows you to maintain, an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns which would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse.

The audit facility acts at an instance level, recording all instance level activities and database level activities.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more than one database partition. Part of each audit record contains information on the coordinator partition and originating database partition identifiers.

The audit log (db2audit.log) and the audit configuration file (db2audit.cfg) are located in the instance's `security` subdirectory. At the time you create an instance, read/write permissions are set on these files, where possible, by the operating system. By default, the permissions are read/write for the instance owner only. It is recommended that you do not change these permissions.

Users of the audit facility administrator tool, db2audit, must have SYSADM authority.

The audit facility must be stopped and started explicitly. When starting, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 database server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log.

Authorized users of the audit facility can control the following actions within the audit facility:

- Start recording auditable events within the DB2 database instance.
- Stop recording auditable events within the DB2 database instance.

- Configure the behavior of the audit facility, including selecting the categories of the auditable events to be recorded.
- Request a description of the current audit configuration.
- Flush any pending audit records from the instance and write them to the audit log.
- Extract audit records by formatting and copying them from the audit log to a flat file or ASCII delimited files. Extraction is done for one of two reasons: in preparation for analysis of log records or in preparation for pruning of log records.
- Prune audit records from the current audit log.

Note: Ensure that the audit facility has been turned on by issuing the db2audit start command before using the audit utilities.

There are different categories of audit records that may be generated. In the description of the categories of events available for auditing (below), you should notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- Audit (AUDIT). Generates records when audit settings are changed or when the audit log is accessed.
- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate DB2 database objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects.
- Security Maintenance (SECMAINT). Generates records when granting or revoking: object or database privileges, or DBADM authority. Records are also generated when the database manager security configuration parameters SYSADM_GROUP, SYSCTRL_GROUP, or SYSMAINT_GROUP are modified.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, a query statement for dynamic queries, a package identifier for static queries, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results.

  Note: The SQL or XQuery statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.
- You can audit failures, successes, or both.

Any operation on the database may generate several records. The actual number of records generated and moved to the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

Related concepts:

- "Audit facility behavior" on page 199

- “Audit facility record layouts (introduction)” on page 212
- “Audit facility tips and techniques” on page 230

Related tasks:

- “Controlling DB2 database audit facility activities” on page 231

Related reference:

- “Audit facility messages” on page 212
- “Audit facility usage” on page 200

Audit facility behavior

The audit facility records auditable events including those affecting database instances. For this reason, the audit facility is an independent part of DB2 database that can operate even if the DB2 database instance is stopped. If the audit facility is active, then when a stopped instance is started, auditing of database events in the instance resumes.

The timing of the writing of audit records to the audit log can have a significant impact on the performance of databases in the instance. The writing of the audit records can take place synchronously or asynchronously with the occurrence of the events causing the generation of those records. The value of the *audit_buf_sz* database manager configuration parameter determines when the writing of audit records is done.

If the value of this parameter is zero (0), the writing is done synchronously. The event generating the audit record will wait until the record is written to disk. The wait associated with each record causes the performance of DB2 database to decrease.

If the value of *audit_buf_sz* is greater than zero, the record writing is done asynchronously. The value of the *audit_buf_sz* when it is greater than zero is the number of 4 KB pages used to create an internal buffer. The internal buffer is used to keep a number of audit records before writing a group of them out to disk. The statement generating the audit record as a result of an audit event will not wait until the record is written to disk, and can continue its operation.

In the asynchronous case, it could be possible for audit records to remain in an unfilled buffer for some time. To prevent this from happening for an extended period, the database manager will force the writing of the audit records regularly. An authorized user of the audit facility may also flush the audit buffer with an explicit request.

There are differences when an error occurs dependent on whether there is synchronous or asynchronous record writing. In asynchronous mode there may be some records lost because the audit records are buffered before being written to disk. In synchronous mode there may be one record lost because the error could only prevent at most one audit record from being written.

The setting of the ERRORTYPE audit facility parameter controls how errors are managed between DB2 database and the audit facility. When the audit facility is active, and the setting of the ERRORTYPE audit facility parameter is `AUDIT`, then the audit facility is treated in the same way as any other part of DB2 database. An audit record must be written (to disk in synchronous mode; or to the audit buffer in asynchronous mode) for an audit event associated with a statement to be considered successful. Whenever an error is encountered when running in this mode, a negative SQLCODE is returned to the application for the statement generating an audit record. If the error type is set to `NORMAL`, then any error from db2audit is ignored and the operation's SQLCODE is returned.

Depending on the API or query statement and the audit settings for the DB2 database instance, none, one, or several audit records may be generated for a particular event. For example, an SQL UPDATE statement with a SELECT subquery may result in one audit record containing the results of the authorization check for UPDATE privilege on a table and another record containing the results of the authorization check for SELECT privilege on a table.

For dynamic data manipulation language (DML) statements, audit records are generated for all authorization checking at the time that the statement is prepared. Reuse of those statements by the same user will not be audited again since no authorization checking takes place at that time. However, if a change has been made to one of the catalog tables containing privilege information, then in the next unit of work, the statement privileges for the cached dynamic SQL or XQuery statements are checked again and one or more new audit records created.

For a package containing only static DML statements, the only auditable event that could generate an audit record is the authorization check to see if a user has the privilege to execute that package. The authorization checking and possible audit record creation required for the static SQL or XQuery statements in the package is carried out at the time the package is precompiled or bound. The execution of the static SQL or XQuery statements within the package is not auditable. When a package is bound again either explicitly by the user, or implicitly by the system, audit records are generated for the authorization checks required by the static SQL or XQuery statements.

For statements where authorization checking is performed at statement execution time (for example, data definition language (DDL), GRANT, and REVOKE statements), audit records are generated whenever these statements are used.

Note: When executing DDL, the section number recorded for all events (except the context events) in the audit record will be zero (0) no matter what the actual section number of the statement might have been.

Related concepts:

- "Introduction to the DB2 database audit facility" on page 197

Related reference:

- "audit_buf_sz - Audit buffer size " on page 1496
- "Audit facility usage" on page 200

Audit facility usage

A review of each part of the following syntax diagrams will assist you in the understanding of how the audit facility can be used.

```
>>-db2audit--+-configure--+-reset-----------------------+-+----------><
             |            '-| Audit Configuration |-----' |
             +-describe-----------------------------------+
             +-extract--| Audit Extraction |--------------+
             +-flush--------------------------------------+
             +-prune--+-all----------------------------------+-+
             |        '-date--YYYYMMDDHH--+---------------------------------+-' |
             |                            '-pathname--Path_with_temp_space--' |
             +-start--------------------------------------+
             '-stop---------------------------------------'
```

Audit Configuration:

```
|--+------------------------+--+----------------------+------------->
   '-scope--+-all-------+---'  '-status--+-both----+--'
            |  .-,---.  |                +-success-+
            |  V     |  |                '-failure-'
            '----+-audit----+-'
                 +-checking-+
                 +-objmaint-+
                 +-secmaint-+
                 +-sysadmin-+
                 +-validate-+
                 '-context--'

>--+-------------------------+---------------------------------------|
   '-errortype--+-audit--+---'
                '-normal-'
```

Audit Extraction:

```
|--+-file--output-file-------------------------+--+-------------------------+-->
   '-delasc--+------------------------------+--'  |          .-,--------.   |
             '-delimiter--load-delimiter----'     |          V          |   |
                                                  '-category----+-audit----+--'
                                                                +-checking-+
                                                                +-objmaint-+
                                                                +-secmaint-+
                                                                +-sysadmin-+
                                                                +-validate-+
                                                                '-context--'

>--+-------------------------+--+-------------------------+----------|
   '-database--database-name-'  '-status--+-success-+-----'
                                          '-failure-'
```

The following is a description and the implied use of each parameter:

configure

This parameter allows the modification of the db2audit.cfg configuration file in the instance's security subdirectory. Updates to this file can occur even when the instance is shut down. Updates occurring when the instance is active dynamically affect the auditing being done by DB2 database across all database partitions. The configure action on the configuration file causes the creation of an audit record if the audit facility has been started and the *audit* category of auditable events is being audited.

The following are the possible actions on the configuration file:

- RESET. This action causes the configuration file to revert to the initial configuration (where SCOPE is all of the categories except CONTEXT, STATUS is FAILURE, ERRORTYPE is NORMAL, and the audit facility is OFF). This action will create a new audit configuration file if the original has been lost or damaged.
- SCOPE. This action specifies which category or categories of events are to be audited. This action also allows a particular focus for auditing and reduces the growth of the log. It is recommended that the number and type of events being logged be limited as much as possible, otherwise the audit log will grow rapidly.

  Note: Please notice that the default SCOPE is all categories except CONTEXT and may result in records being generated rapidly. In conjunction with the mode (synchronous or asynchronous), the selection of the categories may result in a significant performance reduction and significantly increased disk requirements.
- STATUS. This action specifies whether only successful or failing events, or both successful and failing events, should be logged.

  Note: Context events occur before the status of an operation is known. Therefore, such events are logged regardless of the value associated with this parameter.
- ERRORTYPE. This action specifies whether audit errors are returned to the user or are ignored. The value for this parameter can be:
  - AUDIT. All errors including errors occurring within the audit facility are managed by DB2 database and all negative SQLCODEs are reported back to the caller.
  - NORMAL. Any errors generated by db2audit are ignored and only the SQLCODEs for the errors associated with the operation being performed are returned to the application.

describe
This parameter displays to standard output the current audit configuration information and status.

extract This parameter allows the movement of audit records from the audit log to an indicated destination. If no optional clauses are specified, all of the audit records are extracted and placed in a flat report file. If *output_file* already exists, an error message is returned.

The following are the possible options that can be used when extracting:
- FILE. The extracted audit records are placed in a file (*output_file*). If no file name is specified, records are written to the db2audit.out file in the security subdirectory of sqllib. If no directory is specified, *output_file* is written to the current working directory.
- DELASC. The extracted audit records are placed in a delimited ASCII format suitable for loading into DB2 database relational tables. The output is placed in separate files: one for each category. The filenames are:
  - audit.del
  - checking.del
  - objmaint.del
  - secmaint.del
  - sysadmin.del
  - validate.del
  - context.del

These files are always written to the security subdirectory of sqllib.

The DELASC choice also allows you to override the default audit character string delimiter (“0xff”) when extracting from the audit log. You would use DELASC DELIMITER followed by the new delimiter that you wish to use in preparation for loading into a table that will hold the audit records. The new load delimiter can be either a single character (such as !) or a four-byte string representing a hexadecimal number (such as 0xff).

- CATEGORY. The audit records for the specified categories of audit events are to be extracted. If not specified, all categories are eligible for extraction.
- DATABASE. The audit records for a specified database are to be extracted. If not specified, all databases are eligible for extraction.
- STATUS. The audit records for the specified status are to be extracted. If not specified, all records are eligible for extraction.

flush This parameter forces any pending audit records to be written to the audit log. Also, the audit state is reset in the engine from “unable to log” to a state of “ready to log” if the audit facility is in an error state.

prune This parameter allows for the deletion of audit records from the audit log. If the audit facility is active and the “audit” category of events has been specified for auditing, then an audit record will be logged after the audit log is pruned.

The following are the possible options that can be used when pruning:

- ALL. All of the audit records in the audit log are to be deleted.
- DATE yyyymmddhh. The user can specify that all audit records that occurred on or before the date/time specified are to be deleted from the audit log. The user may optionally supply a

  ```
  pathname
  ```

  which the audit facility will use as a temporary space when pruning the audit log. This temporary space allows for the pruning of the audit log when the disk it resides on is full and does not have enough space to allow for a pruning operation.

start This parameter causes the audit facility to begin auditing events based on the contents of the db2audit.cfg file. In a partitioned DB2 database instance, auditing will begin on all database partitions when this clause is specified. If the “audit” category of events has been specified for auditing, then an audit record will be logged when the audit facility is started.

stop This parameter causes the audit facility to stop auditing events. In a partitioned DB2 database instance, auditing will be stopped on all database partitions when this clause is specified. If the “audit” category of events has been specified for auditing, then an audit record will be logged when the audit facility is stopped.

Related concepts:

- “Audit facility tips and techniques” on page 230
- “Introduction to the DB2 database audit facility” on page 197

Related reference:

- “db2audit - Audit facility administrator tool ” on page 474

Working with DB2 audit data in DB2 tables

The following topics describe how to create DB2 audit data, how to create tables to hold this data, how to populate the tables with the DB2 audit data, and how to select the DB2 audit data from the tables.

Working with DB2 audit data in DB2 tables

When you use the DB2 audit facility to maintain an audit trail of database activities, by default the audit facility places the audit records in a log file. If you want, you can write the audit records from the log file to a text file, or you can write the audit records from the log file to delimited ASCII files, then load the contents of the ASCII files into DB2 tables. When the audit data is in DB2 tables, you can select the data from the tables to answer questions that you may have about activity on your DB2 instance.

Procedure:

To work with audit data in DB2 tables:
1. Create tables to hold the DB2 audit data .
2. Create the DB2 audit data files .
3. Use the load utility to populate the tables with the data .
4. Select the table data Select the table data.

Related concepts:
- "Audit facility behavior" on page 199
- "Audit facility tips and techniques" on page 230

Related tasks:
- "Creating tables to hold the DB2 audit data" on page 204
- "Creating DB2 audit data files" on page 207
- "Loading DB2 audit data into tables" on page 208
- "Selecting DB2 audit data from tables" on page 211

Related reference:
- "Audit facility usage" on page 200

Creating tables to hold the DB2 audit data

Before you can work with audit data in tables, you need to create the tables to hold the data. You should consider creating these tables in a separate schema to isolate the data in the tables from unauthorized users.

Prerequisites:
- See the CREATE SCHEMA statement for the authorities and privileges that you require to create a schema.
- See the CREATE TABLE statement for the authorities and privileges that you require to create a table.
- Decide which table space you want to use to hold the tables. (This topic does not describe how to create table spaces.)

Procedure:

The examples that follow show how to the create tables that will hold all of the records from all of the ASCII files. If you want, you can create a separate schema to contain these tables.

If you do not want to use all of the data that is contained in the files, you can omit columns from the table definitions, or bypass creating tables, as required. If you omit columns from the table definitions, you must modify the commands that you use to load data into these tables.

1. Issue the db2 command to open a DB2 command window.
2. Optional. Create a schema to hold the tables. Issue the following command. For this example, the schema is called AUDIT
   ```
   CREATE SCHEMA AUDIT
   ```
3. Optional. If you created the AUDIT schema, switch to the schema before creating any tables. Issue the following command:
   ```
   SET CURRENT SCHEMA = 'AUDIT'
   ```
4. To create the table that will contain records from the audit.del file, issue the following SQL statement:
   ```
   CREATE TABLE AUDIT (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128))
   ```
5. To create the table that will contain records from the checking.del file, issue the following SQL statement:
   ```
   CREATE TABLE CHECKING (TIMESTAMP CHAR(26),
                          CATEGORY CHAR(8),
                          EVENT VARCHAR(32),
                          CORRELATOR INTEGER,
                          STATUS INTEGER,
                          DATABASE CHAR(8),
                          USERID VARCHAR(1024),
                          AUTHID VARCHAR(128),
                          NODENUM SMALLINT,
                          COORDNUM SMALLINT,
                          APPID VARCHAR(255),
                          APPNAME VARCHAR(1024),
                          PKGSCHEMA VARCHAR(128),
                          PKGNAME VARCHAR(128),
                          PKGSECNUM SMALLINT,
                          OBJSCHEMA VARCHAR(128),
                          OBJNAME VARCHAR(128),
                          OBJTYPE VARCHAR(32),
                          ACCESSAPP CHAR(18),
                          ACCESSATT CHAR(18),
                          PKGVER VARCHAR(64),
                          CHKAUTHID VARCHAR(128))
   ```
6. To create the table that will contain records from the objmaint.del file, issue the following SQL statement:
   ```
   CREATE TABLE OBJMAINT (TIMESTAMP CHAR(26),
                          CATEGORY CHAR(8),
                          EVENT VARCHAR(32),
                          CORRELATOR INTEGER,
                          STATUS INTEGER,
                          DATABASE CHAR(8),
                          USERID VARCHAR(1024),
                          AUTHID VARCHAR(128),
                          NODENUM SMALLINT,
                          COORDNUM SMALLINT,
   ```

```
                        APPID VARCHAR(255),
                        APPNAME VARCHAR(1024),
                        PKGSCHEMA VARCHAR(128),
                        PKGNAME VARCHAR(128),
                        PKGSECNUM SMALLINT,
                        OBJSCHEMA VARCHAR(128),
                        OBJNAME VARCHAR(128),
                        OBJTYPE VARCHAR(32),
                        PACKVER VARCHAR(64))
```

7. To create the table that will contain records from the secmaint.del file, issue the following SQL statement:

```
CREATE TABLE SECMAINT (TIMESTAMP CHAR(26),
                        CATEGORY CHAR(8),
                        EVENT VARCHAR(32),
                        CORRELATOR INTEGER,
                        STATUS INTEGER,
                        DATABASE CHAR(8),
                        USERID VARCHAR(1024),
                        AUTHID VARCHAR(128),
                        NODENUM SMALLINT,
                        COORDNUM SMALLINT,
                        APPID VARCHAR(255),
                        APPNAME VARCHAR(1024),
                        PKGSCHEMA VARCHAR(128),
                        PKGNAME VARCHAR(128),
                        PKGSECNUM SMALLINT,
                        OBJSCHEMA VARCHAR(128),
                        OBJNAME VARCHAR(128),
                        OBJTYPE VARCHAR(32),
                        GRANTOR VARCHAR(128),
                        GRANTEE VARCHAR(128),
                        GRANTEETYPE VARCHAR(32),
                        PRIVAUTH CHAR(18),
                        PKGVER VARCHAR(64))
```

8. To create the table that will contain records from the sysadmin.del file, issue the following SQL statement:

```
CREATE TABLE SYSADMIN (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       DATABASE CHAR(8),
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128),
                       NODENUM SMALLINT,
                       COORDNUM SMALLINT,
                       APPID VARCHAR(255),
                       APPNAME VARCHAR(1024),
                       PKGSCHEMA VARCHAR(128),
                       PKGNAME VARCHAR(128),
                       PKGSECNUM SMALLINT,
                       PKGVER VARCHAR(64))
```

9. To create the table that will contain records from the validate.del file, issue the following SQL statement:

```
CREATE TABLE VALIDATE (TIMESTAMP CHAR(26),
                       CATEGORY CHAR(8),
                       EVENT VARCHAR(32),
                       CORRELATOR INTEGER,
                       STATUS INTEGER,
                       DATABASE CHAR(8),
                       USERID VARCHAR(1024),
                       AUTHID VARCHAR(128),
                       EXECID VARCHAR(1024),
                       NODENUM SMALLINT,
                       COORDNUM SMALLINT,
                       APPID VARCHAR(255),
                       APPNAME VARCHAR(1024),
                       AUTHTYPE VARCHAR(32),
                       PKGSCHEMA VARCHAR(128),
                       PKGNAME VARCHAR(128),
                       PKGSECNUM SMALLINT,
                       PKGVER VARCHAR(64),
                       PLUGINNAME VARCHAR(32))
```

10. To create the table that will contain records from the context.del file, issue the following SQL statement:

```
    CREATE TABLE CONTEXT (TIMESTAMP CHAR(26),
                          CATEGORY CHAR(8),
                          EVENT VARCHAR(32),
                          CORRELATOR INTEGER,
                          DATABASE CHAR(8),
                          USERID VARCHAR(1024),
                          AUTHID VARCHAR(128),
                          NODENUM SMALLINT,
                          COORDNUM SMALLINT,
                          APPID VARCHAR(255),
                          APPNAME VARCHAR(1024),
                          PKGSCHEMA VARCHAR(128),
                          PKGNAME VARCHAR(128),
                          PKGSECNUM SMALLINT,
                          STMTTEXT CLOB(2M),
                          PKGVER VARCHAR(64))
```

11. After creating the tables, issue the COMMIT statement to ensure that the table definitions are written to disk.
12. When you have created the tables, you are ready to extract the audit records from the db2audit.log file to delimited ASCII files.

Related tasks:

- "Creating DB2 audit data files" on page 207
- "Setting a schema" on page 305

Related reference:

- "CREATE SCHEMA " on page 948
- "CREATE TABLE " on page 956

Creating DB2 audit data files

By default, the DB2 audit facility writes audit data to the db2audit.log file. The records in this file cannot be loaded into tables. You must extract the audit records to delimited ASCII files, which can you use to populate tables.

Prerequisites:

You require SYSADM authority to use the `db2audit` command.

Procedure:

To write the audit facility records to delimited ASCII files:

1. Review the topic on audit facility usage to determine the type of DB2 activities that you want to audit. When you are satisfied with the configuration that you have set up for the audit facility, issue the following command to begin auditing:

```
db2audit start
```

2. Issue the following command to ensure that all audit records are flushed from memory to the db2audit.log file:

   ```
   db2audit flush
   ```

3. Issue the following command to move the audit records from the db2audit.log to delimited ASCII files:

   ```
   db2audit extract delasc
   ```

   The following files are created in the security subdirectory of sqllib. If you are not auditing a particular type of event, the file for that event is created, but the file is empty.
   - audit.del
   - checking.del
   - objmaint.del
   - secmaint.del
   - sysadmin.del
   - validate.del
   - context.del
4. Issue the following command to delete the audit records from the db2audit.log file that you just extracted:

   ```
   db2audit prune date YYYYMMDDHH
   ```

   Where *YYYYMMDDHH* is the current year, month, day, and hour. Write down the value that you use because you will require this information in the next step when you populate the tables with the audit data.

   The audit facility will continue to write new audit records to the db2audit.log file, and these records will have a timestamp that is later than *YYYYMMDDHH*. Pruning records from the db2audit.log file that you have already extracted prevents you from extracting the same records a second time. All audit records that are written after *YYYYMMDDHH* will be written to the .del files the next time you extract the audit data.
5. After you create the audit data files, the next step is to use the load utility to populate the tables with the audit data.

Related tasks:
- "Loading DB2 audit data into tables" on page 208

Related reference:
- "Audit record layout for SYSADMIN events" on page 226
- "Audit record layout for VALIDATE events" on page 227
- "db2audit - Audit facility administrator tool " on page 474
- "Audit facility usage" on page 200
- "Audit record layout for AUDIT events" on page 213
- "Audit record layout for CHECKING events" on page 214
- "Audit record layout for CONTEXT events" on page 228
- "Audit record layout for OBJMAINT events" on page 219
- "Audit record layout for SECMAINT events" on page 221

Loading DB2 audit data into tables

When you have created the tables to hold the audit data, you then load the data in the ASCII files into the tables.

Prerequisites:

See the topic on the privileges, authorities, and authorizations required to use the load utility for more information.

Procedure:

Use the load utility to load the data into the tables. Issue a separate load command for each table. If you omitted one or more columns from the table definitions, you must modify the version of the LOAD command that you use to successfully load the data. Also, if you specified a delimiter character other than the default (`0xff`) when you extracted the audit data, you must also modify the version of the LOAD command that you use (see the topic ″File type modifiers for load ″ for more information).

1. Issue the `db2` command to open a DB2 command window.
2. To load the AUDIT table, issue the following command:

   ```
   LOAD FROM audit.del OF del MODIFIED BY CHARDEL0xff INSERT INTO schema.AUDIT
   ```

   Note: When specifying the file name, usen the fully qualified path name. For example, if you have DB2 database installed on the C: drive of a Windows-based computer, you would specify `C:\Program Files\IBM\SQLLIB\instance\security\audit.del` as the fully qualified file name for the audit.del file.

   After loading the AUDIT table, issue the following DELETE statement to ensure that you do not load duplicate rows into the table the next time you load it. When you extracted the audit records from the db2audit.log file, all records in the file were written to the .del files. Likely, the .del files contained records that were written after the hour to which the audit log was subsequently pruned (because the `db2audit prune` command only prunes records to a specified hour). The next time you extract the audit records, the new .del files will contain records that were previously extracted, but not deleted by the `db2audit prune` command (because they were written after the hour specified for the prune operation). Deleting rows from the table to the same hour to which the db2audit.log file was pruned ensures that the table does not contain duplicate rows, and that no audit records are lost.

   ```
   DELETE FROM schema.AUDIT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the db2audit.log file. Because the DB2 audit facility continues to write audit records to the db2audit.log file after it is pruned, you must specify `0000` for the minutes and seconds to ensure that audit records that were written after the db2audit.log file was pruned are not deleted from the table.
3. To load the CHECKING table, issue the following command:

   ```
   LOAD FROM checking.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
      schema.CHECKING
   ```

   After loading the CHECKING table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

   ```
   DELETE FROM schema.CHECKING WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
   ```

   Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.
4. To load the OBJMAINT table, issue the following command:

```
LOAD FROM objmaint.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
   schema.OBJMAINT
```

After loading the OBJMAINT table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.OBJMAINT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

5. To load the SECMAINT table, issue the following command:

```
LOAD FROM secmaint.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
   schema.SECMAINT
```

After loading the SECMAINT table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.SECMAINT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

6. To load the SYSADMIN table, issue the following command:

```
LOAD FROM sysadmin.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
   schema.SYSADMIN
```

After loading the SYSADMIN table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.SYSADMIN WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

7. To load the VALIDATE table, issue the following command:

```
LOAD FROM validate.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
   schema.VALIDATE
```

After loading the VALIDATE table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.VALIDATE WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

8. To load the CONTEXT table, issue the following command:

```
LOAD FROM context.del OF del MODIFIED BY CHARDEL0xff INSERT INTO
   schema.CONTEXT
```

After loading the CONTEXT table, issue the following SQL statement to ensure that you do not load duplicate rows into the table the next time you load it:

```
DELETE FROM schema.CONTEXT WHERE TIMESTAMP > TIMESTAMP('YYYYMMDDHH0000')
```

Where *YYYYMMDDHH* is the value that you specified when you pruned the log file.

9. After you finish loading the data into the tables, delete the .del files from the security subdirectory of the sqllib directory.
10. When you have loaded the audit data into the tables, you are ready to select data from these tablesselect data from these tables.

If you have already populated the tables a first time, and want to do so again, use the INSERT option to have the new table data added to the existing table data. If you want to have the records from the previous `db2audit extract` operation removed from the tables, load the tables again using the REPLACE option. In either situation, remember both to flush the audit records to the db2audit.log file before extracting the records to the .del files, and to prune the db2audit.log file after extracting the records so that you do not load the same records into the tables more than once.

Related concepts:

- "Privileges, authorities, and authorizations required to use Load" on page 414
- "" in *Administration Guide: Planning*
- "LOAD authority" on page 77

Related tasks:

- "Selecting DB2 audit data from tables" on page 211
- "Enabling parallelism for loading data" on page 53
- "Loading data into a table using the Load wizard" in *Administration Guide: Implementation*

Related reference:

- "File type modifiers for the load utility" on page 555

Selecting DB2 audit data from tables

When the audit data is successfully loaded into the tables, you can select data from these tables for further analysis.

Prerequisites:

See the topic on the SELECT statement for information about the authorities and privileges required to select data from a table.

Procedure:

To select all the rows in a table:

1. Issue the `db2` command to open a DB2 command window.
2. Issue an SQL statement of the following form for each table from which you want to select audit data:
   ```
   SELECT * FROM SQL schema.table
   ```

For example, to select all the data from the `CHECKING` table in the `AUDIT` schema, use the following statement:

```
SELECT * FROM AUDIT.CHECKING
```

The select that you perform should reflect the type of analysis that you want to do on the data. For example, you can select records according to an authorization ID (authid) to determine the type of activities that this authorization ID has been performing:

```
SELECT * FROM AUDIT.CHECKING WHERE AUTHID = authorization ID
```

Where *authorization ID* is the user ID for which you want to analyze the data.

For a description of the values that can be included in audit data, see the corresponding audit record layout topic for the table, and the list of possible returned values for the table.

Related reference:
- “Subselect” on page 1211
- “SELECT ” on page 1182
- “Audit record layout for AUDIT events” on page 213
- “Audit record layout for CHECKING events” on page 214
- “Audit record layout for CONTEXT events” on page 228
- “Audit record layout for OBJMAINT events” on page 219
- “Audit record layout for SECMAINT events” on page 221
- “Audit record layout for SYSADMIN events” on page 226
- “Audit record layout for VALIDATE events” on page 227
- “List of possible CHECKING access approval reasons” on page 216
- “List of possible CHECKING access attempted types” on page 217
- “List of possible CONTEXT audit events” on page 229
- “List of possible SECMAINT privileges or authorities” on page 223
- “List of possible SYSADMIN audit events” on page 226

Audit facility messages

SQL1322N An error occurred when writing to the audit log file.

Explanation: The DB2 database audit facility encountered an error when invoked to record an audit event to the audit log file. There is no space on the file system where the audit log resides.

User response: The system administrator should free up space on this file system or prune the audit log to reduce its size.

When more space is available, use db2audit to flush out any data in memory, and to reset the auditor to a ready state. Ensure that appropriate extracts have occurred, or a copy of the log has been made before pruning the log, as deleted records are not recoverable.

sqlcode: -1322

sqlstate: 50830

SQL1323N An error occurred when accessing the audit configuration file.

Explanation: The audit configuration file (db2audit.cfg) could not be opened, or was invalid. Possible reasons for this error are that the db2audit.cfg file either does not exist, or has been damaged.

User response: Take one of the following actions:
- Restore from a saved version of the file.
- Reset the audit facility configuration file by issuing

```
db2audit reset
```

sqlcode: -1323

sqlstate: 57019

Related concepts:
- “Introduction to the DB2 database audit facility” on page 197

Audit facility record layouts (introduction)

When an audit record is extracted from the audit log using the DELASC extract option, each record will have one of the formats shown in the following tables. Each table will begin by showing the contents of a sample record. The description of each item of the record is shown one row at a time in the associated table. If the item is important, the name of the item will be highlighted (**bold**). These items contain information that are of most interest to you.

Notes:

1. Not all fields in the sample records will have values.
2. Some fields such as "Access Attempted" are stored in the delimited ASCII format as bitmaps. In this flat report file, however, these fields will appear as a set of strings representing the bitmap values.
3. A new field called "Package Version" has been added to the record layout for the CHECKING, OBJMAINT, SECMAINT, SYSADMIN, VALIDATE, and CONTEXT events.

Related reference:

- "Audit record layout for AUDIT events" on page 213
- "Audit record layout for CHECKING events" on page 214
- "Audit record layout for CONTEXT events" on page 228
- "Audit record layout for OBJMAINT events" on page 219
- "Audit record layout for SECMAINT events" on page 221
- "Audit record layout for SYSADMIN events" on page 226
- "Audit record layout for VALIDATE events" on page 227

Details on audit facility record layouts

The various audit facility record layouts are shown in this section.

Audit record layout for AUDIT events

Sample audit record:

```
timestamp=1998-06-24-11.54.05.151232;category=AUDIT;audit event=START;
  event correlator=0;event status=0;
  userid=boss;authid=BOSS;
```

Table 58. Audit Record Layout for AUDIT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: AUDIT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: CONFIGURE, DB2AUD, EXTRACT, FLUSH, PRUNE, START, STOP, and UPDATE_ADMIN_CFG
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.

Related concepts:

- "Audit facility record layouts (introduction)" on page 212

Audit record layout for CHECKING events

Sample audit record:

```
timestamp=1998-06-24-08.42.11.622984;category=CHECKING;audit event=CHECKING_OBJECT;
  event correlator=2;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  package schema=NULLID;package name=SYSSH200;
  package section=0;object schema=GSTAGER;object name=NONE;object type=REOPT_VALUES;
  access approval reason=DBADM;access attempted=STORE;
```

Table 59. Audit record layout for CHECKING events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: CHECKING
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: CHECKING_OBJECT, CHECKING_FUNCTION, and CHECKING_TRANSFER
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR (128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR (128)	Name of object for which the audit event was generated.
Object Type	VARCHAR (32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled "Audit record object types".
Access Approval Reason	CHAR(18)	Indicates the reason why access was approved for this audit event. Possible values include: those shown in the topic titled "List of possible CHECKING access approval reasons".
Access Attempted	CHAR(18)	Indicates the type of access that was attempted. Possible values include: those shown in the topic titled "List of possible CHECKING access attempted types".

Table 59. Audit record layout for CHECKING events (continued)

NAME	FORMAT	DESCRIPTION
Package Version	VARCHAR (64)	Version of the package in use at the time that the audit event occurred.
Checked Authorization ID	VARCHAR(128)	Authorization ID is checked when it is different than the authorization ID at the time of the audit event. For example, this can be the target owner in a TRANSFER OWNERSHIP statement.

Related concepts:
- "Audit facility record layouts (introduction)" on page 212

Related reference:
- "Audit record object types" on page 215
- "List of possible CHECKING access approval reasons" on page 216
- "List of possible CHECKING access attempted types" on page 217

Audit record object types

Table 60. Audit Record Object Types Based on Audit Events

Object type	CHECKING events	OBJMAINT events	SECMAINT events
NONE	X	X	X
TABLE	X	X	X
VIEW	X	X	X
ALIAS	X	X	
FUNCTION	X	X	X
INDEX	X	X	X
INDEX EXTENSION		X	
PACKAGE	X	X	X
PACKAGE CACHE	X		
DATA_TYPE		X	
NODEGROUP	X	X	
SCHEMA	X	X	X
STORED_PROCEDURE	X	X	X
METHOD_BODY	X	X	X
BUFFERPOOL	X	X	
SEQUENCE	X	X	
TABLESPACE	X	X	X
EVENT_MONITOR	X	X	
TRIGGER		X	
DATABASE	X		X
INSTANCE	X		
FOREIGN_KEY		X	
PRIMARY_KEY		X	
UNIQUE_CONSTRAINT		X	

Table 60. Audit Record Object Types Based on Audit Events (continued)

Object type	CHECKING events	OBJMAINT events	SECMAINT events
CHECK_CONSTRAINT		X	
WRAPPER	X	X	
SERVER	X	X	X
NICKNAME	X	X	X
USER MAPPING	X	X	
SERVER OPTION	X	X	
TYPE&TRANSFORM	X	X	
TYPE MAPPING	X	X	
FUNCTION MAPPING	X	X	
SUMMARY TABLES	X	X	X
JAR_FILE		X	
ALL	X		
REOPT_VALUES	X		
SECURITY_LABEL		X	X
SECURITY_POLICY		X	X
SECURITY_LABEL_COMPONENT		X	
ACCESS_RULE			X
XSR object	X	X	X

Related reference:

- "Audit record layout for CHECKING events" on page 214
- "Audit record layout for OBJMAINT events" on page 219
- "Audit record layout for SECMAINT events" on page 221

List of possible CHECKING access approval reasons

The following is the list of possible CHECKING access approval reasons:

0x0000000000000001 ACCESS DENIED
Access is not approved; rather, it was denied.

0x0000000000000002 SYSADM
Access is approved; the application or user has SYSADM authority.

0x0000000000000004 SYSCTRL
Access is approved; the application or user has SYSCTRL authority.

0x0000000000000008 SYSMAINT
Access is approved; the application or user has SYSMAINT authority.

0x0000000000000010 DBADM
Access is approved; the application or user has DBADM authority.

0x0000000000000020 DATABASE PRIVILEGE
Access is approved; the application or user has an explicit privilege on the database.

0x0000000000000040 OBJECT PRIVILEGE
: Access is approved; the application or user has an explicit privilege on the object or function.

0x0000000000000080 DEFINER
: Access is approved; the application or user is the definer of the object or function.

0x0000000000000100 OWNER
: Access is approved; the application or user is the owner of the object or function.

0x0000000000000200 CONTROL
: Access is approved; the application or user has CONTROL privilege on the object or function.

0x0000000000000400 BIND
: Access is approved; the application or user has bind privilege on the package.

0x0000000000000800 SYSQUIESCE
: Access is approved; if the instance or database is in quiesce mode, the application or user may connect or attach.

0x0000000000001000 SYSMON
: Access is approved; the application or user has SYSMON authority.

0x0000000000002000 SECADM
: Access is approved; the application or user has SECADM authority.

0x0000000000004000 SETSESSIONUSER
: Access is approved; the application or user has SETSESSIONUSER authority.

Related reference:
- "Audit record layout for CHECKING events" on page 214
- "List of possible CHECKING access attempted types" on page 217

List of possible CHECKING access attempted types

The following is the list of possible CHECKING access attempted types. If Audit Event is CHECKING_TRANSFER, then the audit entry reflects that a privilege is held or not.

0x0000000000000001 CONTROL
: Attempt to verify if CONTROL privilege is held.

0x0000000000000002 ALTER
: Attempt to alter an object or to verify if ALTER privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000004 DELETE
: Attempt to delete an object or to verify if DELETE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000008 INDEX
: Attempt to use an index or to verify if INDEX privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000010 INSERT
: Attempt to insert into an object or to verify if INSERT privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000020 SELECT
Attempt to query a table or view or to verify if SELECT privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000040 UPDATE
Attempt to update data in an object or to verify if UPDATE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000080 REFERENCE
Attempt to establish referential constraints between objects or to verify if REFERENCE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000000100 CREATE
Attempt to create an object.

0x0000000000000200 DROP
Attempt to drop an object.

0x0000000000000400 CREATEIN
Attempt to create an object within another schema.

0x0000000000000800 DROPIN
Attempt to drop an object found within another schema.

0x0000000000001000 ALTERIN
Attempt to alter or modify an object found within another schema.

0x0000000000002000 EXECUTE
Attempt to execute or run an application or to invoke a routine, create a function sourced from the routine (applies to functions only), or reference a routine in any DDL statement or to verify if EXECUTE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000000004000 BIND
Attempt to bind or prepare an application.

0x0000000000008000 SET EVENT MONITOR
Attempt to set event monitor switches.

0x0000000000010000 SET CONSTRAINTS
Attempt to set constraints on an object.

0x0000000000020000 COMMENT ON
Attempt to create comments on an object.

0x0000000000040000 GRANT
Attempt to grant privileges on an object to another user ID.

0x0000000000080000 REVOKE
Attempt to revoke privileges on an object from a user ID.

0x0000000000100000 LOCK
Attempt to lock an object.

0x0000000000200000 RENAME
Attempt to rename an object.

0x0000000000400000 CONNECT
Attempt to connect to an object.

0x0000000000800000 Member of SYS Group
Attempt to access or use a member of the SYS group.

0x0000000001000000 Access All
: Attempt to execute a statement with all required privileges on objects held (only used for DBADM/SYSADM).

0x0000000002000000 Drop All
: Attempt to drop multiple objects.

0x0000000004000000 LOAD
: Attempt to load a table in a table space.

0x0000000008000000 USE
: Attempt to create a table in a table space or to verify if USE privilege is held if Audit Event is CHECKING_TRANSFER.

0x0000000010000000 SET SESSION_USER
: Attempt to execute the SET SESSION_USER statement.

0x0000000020000000 FLUSH
: Attempt to execute the FLUSH statement.

0x0000000040000000 STORE
: Attempt to view the values of a re-optimized statement in the EXPLAIN_PREDICATE table.

0x0000000400000000 TRANSFER
: Attempt to transfer an object.

0x0000000800000000 ALTER_WITH_GRANT
: Attempt to verify if ALTER with GRANT privilege is held.

0x0000001000000000 DELETE_WITH_GRANT
: Attempt to verify if DELETE with GRANT privilege is held.

0x0000002000000000 INDEX_WITH_GRANT
: Attempt to verify if INDEX with GRANT privilege is held

0x0000004000000000 INSERT_WITH_GRANT
: Attempt to verify if INSERT with GRANT privilege is held.

0x0000008000000000 SELECT_WITH_GRANT
: Attempt to verify if SELECT with GRANT privilege is held.

0x0000010000000000 UPDATE_WITH_GRANT
: Attempt to verify if UPDATE with GRANT privilege is held.

0x0000020000000000 REFERENCE_WITH_GRANT
: Attempt to verify if REFERENCE with GRANT privilege is held.

0x0000040000000000 USAGE
: Attempt to use a sequence or an XSR object or to verify if USAGE privilege is held if Audit Event is CHECKING_TRANSFER.

Related reference:
- "Audit record layout for CHECKING events" on page 214
- "List of possible CHECKING access approval reasons" on page 216

Audit record layout for OBJMAINT events

Sample audit record:

```
timestamp=1998-06-24-08.42.41.957524;category=OBJMAINT;audit event=CREATE_OBJECT;
  event correlator=3;event status=0;
  database=FOO;userid=boss;authid=BOSS;
```

```
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;
package section=0;object schema=BOSS;object name=AUDIT;object type=TABLE;
```

Table 61. Audit Record Layout for OBJMAINT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: OBJMAINT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: CREATE_OBJECT, RENAME_OBJECT, DROP_OBJECT, and ALTER_OBJECT. ALTER_OBJECT events are generated only when altering protected tables.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR (128)	Schema of the object for which the audit event was generated.
Object Name	VARCHAR (128)	Name of object for which the audit event was generated.
Object Type	VARCHAR (32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled "Audit record object types".
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.
Security Policy Name	VARCHAR(128)	The name of the security policy if the object type is TABLE and that table is associated with a security policy.

Table 61. Audit Record Layout for OBJMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Alter Action	VARCHAR(32)	Specific Alter operation Possible values include: • ADD_PROTECTED_COLUMN • ADD_COLUMN_PROTECTION • DROP_COLUMN_PROTECTION • ADD_ROW_PROTECTION • ADD_SECURITY_POLICY
Protected Column Name	VARCHAR(128)	If the Alter Action is ADD_COLUMN_PROTECTION or DROP_COLUMN_PROTECTION this is the name of the affected column.
Column Security Label	VARCHAR(128)	The security label protecting the column specified in the field Column Name.
Security Label Column Name	VARCHAR(128)	Name of the column containing the security label protecting the row.

Related concepts:

- "Introduction to the DB2 database audit facility" on page 197

Related reference:

- "Audit record object types" on page 215

Audit record layout for SECMAINT events

Sample audit record:

```
timestamp=1998-06-24-11.57.45.188101;category=SECMAINT;audit event=GRANT;
  event correlator=4;event status=0;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.boss.980624155728;application name=db2bp;
  package schema=NULLID;package name=SQLC28A1;
  package section=0;object schema=BOSS;object name=T1;object type=TABLE;
  grantor=BOSS;grantee=WORKER;grantee type=USER;privilege=SELECT;
```

Table 62. Audit Record Layout for SECMAINT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: SECMAINT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: GRANT, REVOKE, IMPLICIT_GRANT, IMPLICIT_REVOKE, SET_SESSION_USER, UPDATE_DBM_CFG, and TRANSFER_OWNERSHIP.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0

Table 62. Audit Record Layout for SECMAINT Events (continued)

NAME	**FORMAT**	**DESCRIPTION**
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Object Schema	VARCHAR (128)	Schema of the object for which the audit event was generated. If the object type field is ACCESS_RULE then this field contains the security policy name associated with the rule. The name of the rule is stored in the field Object Name. If the object type field is SECURITY_LABEL, then this field contains the name of the security policy that the security label is part of. The name of the security label is stored in the field Object Name.
Object Name	VARCHAR (128)	Name of object for which the audit event was generated. If the object type field is ACCESS_RULE then this field contains the name of the rule. The security policy name associated with the rule is stored in the field Object Schema. If the object type field is SECURITY_LABEL, then this field contains the name of the security label. The name of the security policy that it is part of is stored in the field Object Schema.
Object Type	VARCHAR (32)	Type of object for which the audit event was generated. Possible values include: those shown in the topic titled “Audit record object types”.
Grantor	VARCHAR (128)	Grantor ID.
Grantee	VARCHAR (128)	Grantee ID for which a privilege or authority was granted or revoked.
Grantee Type	VARCHAR (32)	Type of the grantee that was granted to or revoked from. Possible values include: USER, GROUP, or BOTH.
Privilege or Authority	CHAR(18)	Indicates the type of privilege or authority granted or revoked. Possible values include: those shown in the topic titled “List of possible SECMAINT privileges or authorities”.
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.
Access Type	VARCHAR(32)	The access type for which a security label is granted. Possible values: • READ • WRITE • ALL

Table 62. Audit Record Layout for SECMAINT Events (continued)

NAME	FORMAT	DESCRIPTION
Assumable Authid	VARCHAR(128)	When the privilege granted is a SETSESSIONUSER privilege this is the authorization ID that the grantee is allowed to set as the session user.

Related concepts:
- "Audit facility record layouts (introduction)" on page 212

Related reference:
- "Audit record object types" on page 215
- "List of possible SECMAINT privileges or authorities" on page 223

List of possible SECMAINT privileges or authorities

The following is the list of possible SECMAINT privileges or authorities:

0x0000000000000001 Control Table
Control privilege granted or revoked on a table or view.

0x0000000000000002 ALTER TABLE
Privilege granted or revoked to alter a table.

0x0000000000000004 ALTER TABLE with GRANT
Privilege granted or revoked to alter a table with granting of privileges allowed.

0x0000000000000008 DELETE TABLE
Privilege granted or revoked to drop a table or view.

0x0000000000000010 DELETE TABLE with GRANT
Privilege granted or revoked to drop a table with granting of privileges allowed.

0x0000000000000020 Table Index
Privilege granted or revoked on an index.

0x0000000000000040 Table Index with GRANT
Privilege granted or revoked on an index with granting of privileges allowed.

0x0000000000000080 Table INSERT
Privilege granted or revoked on an insert on a table or view.

0x0000000000000100 Table INSERT with GRANT
Privilege granted or revoked on an insert on a table with granting of privileges allowed.

0x0000000000000200 Table SELECT
Privilege granted or revoked on a select on a table.

0x0000000000000400 Table SELECT with GRANT
Privilege granted or revoked on a select on a table with granting of privileges allowed.

0x0000000000000800 Table UPDATE
Privilege granted or revoked on an update on a table or view.

0x0000000000001000 Table UPDATE with GRANT
Privilege granted or revoked on an update on a table or view with granting of privileges allowed.

0x0000000000002000 Table REFERENCE
Privilege granted or revoked on a reference on a table.

0x0000000000004000 Table REFERENCE with GRANT
Privilege granted or revoked on a reference on a table with granting of privileges allowed.

0x0000000000020000 CREATEIN Schema
CREATEIN privilege granted or revoked on a schema.

0x0000000000040000 CREATEIN Schema with GRANT
CREATEIN privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000080000 DROPIN Schema
DROPIN privilege granted or revoked on a schema.

0x0000000000100000 DROPIN Schema with GRANT
DROPIN privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000200000 ALTERIN Schema
ALTERIN privilege granted or revoked on a schema.

0x0000000000400000 ALTERIN Schema with GRANT
ALTERIN privilege granted or revoked on a schema with granting of privileges allowed.

0x0000000000800000 DBADM Authority
DBADM authority granted or revoked.

0x0000000001000000 CREATETAB Authority
Createtab authority granted or revoked.

0x0000000002000000 BINDADD Authority
Bindadd authority granted or revoked.

0x0000000004000000 CONNECT Authority
CONNECT authority granted or revoked.

0x0000000008000000 Create not fenced Authority
Create not fenced authority granted or revoked.

0x0000000010000000 Implicit Schema Authority
Implicit schema authority granted or revoked.

0x0000000020000000 Server PASSTHRU
Privilege granted or revoked to use the pass-through facility with this server (federated database data source).

0x0000000100000000 Table Space USE
Privilege granted or revoked to create a table in a table space.

0x0000000200000000 Table Space USE with GRANT
Privilege granted or revoked to create a table in a table space with granting of privileges allowed.

0x0000000400000000 Column UPDATE
Privilege granted or revoked on an update on one or more specific columns of a table.

0x0000000800000000 Column UPDATE with GRANT
: Privilege granted or revoked on an update on one or more specific columns of a table with granting of privileges allowed.

0x0000001000000000 Column REFERENCE
: Privilege granted or revoked on a reference on one or more specific columns of a table.

0x0000002000000000 Column REFERENCE with GRANT
: Privilege granted or revoked on a reference on one or more specific columns of a table with granting of privileges allowed.

0x0000004000000000 LOAD Authority
: LOAD authority granted or revoked.

0x0000008000000000 Package BIND
: BIND privilege granted or revoked on a package.

0x0000010000000000 Package BIND with GRANT
: BIND privilege granted or revoked on a package with granting of privileges allowed.

0x0000020000000000 EXECUTE
: EXECUTE privilege granted or revoked on a package or a routine.

0x0000040000000000 EXECUTE with GRANT
: EXECUTE privilege granted or revoked on a package or a routine with granting of privileges allowed.

0x0000080000000000 EXECUTE IN SCHEMA
: EXECUTE privilege granted or revoked for all routines in a schema.

0x0000100000000000 EXECUTE IN SCHEMA with GRANT
: EXECUTE privilege granted or revoked for all routines in a schema with granting of privileges allowed.

0x000020000000000 EXECUTE IN TYPE
: EXECUTE privilege granted or revoked for all routines in a type.

0x0000400000000000 EXECUTE IN TYPE with GRANT
: EXECUTE privilege granted or revoked for all routines in a type with granting of privileges allowed.

0x000080000000000 CREATE EXTERNAL ROUTINE
: CREATE EXTERNAL ROUTINE privilege granted or revoked.

0x0001000000000000 QUIESCE_CONNECT
: QUIESCE_CONNECT privilege granted or revoked.

0x0004000000000000 SECADM Authority
: SECADM authority granted or revoked

0x0040000000000000 SETSESSIONUSER Privilege
: SETSESSIONUSER granted or revoked

0x0080000000000000 Exemption
: Exemption granted or revoked

0x0100000000000000 Security label
: Security label granted or revoked

Related reference:

- "Audit record layout for SECMAINT events" on page 221

Audit record layout for SYSADMIN events

Sample audit record:

```
timestamp=1998-06-24-11.54.04.129923;category=SYSADMIN;audit event=DB2AUDIT;
  event correlator=1;event status=0;
  userid=boss;authid=BOSS;
  application id=*LOCAL.boss.980624155404;application name=db2audit;
```

Table 63. Audit Record Layout for SYSADMIN Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: SYSADMIN
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: Those shown in the list following this table.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.

Related concepts:
- "Audit facility record layouts (introduction)" on page 212

Related reference:
- "List of possible SYSADMIN audit events" on page 226

List of possible SYSADMIN audit events

The following is the list of possible SYSADMIN audit events:

Table 64. SYSADMIN Audit Events

START_DB2	ROLLFORWARD_DB
STOP_DB2	SET_RUNTIME_DEGREE
CREATE_DATABASE	SET_TABLESPACE_CONTAINERS
ALTER_DATABASE	UNCATALOG_DB
DROP_DATABASE	UNCATALOG_DCS_DB
UPDATE_DBM_CFG	UNCATALOG_NODE
UPDATE_DB_CFG	UPDATE_ADMIN_CFG
CREATE_TABLESPACE	UPDATE_MON_SWITCHES
DROP_TABLESPACE	LOAD_TABLE
ALTER_TABLESPACE	DB2AUDIT
RENAME_TABLESPACE	SET_APPL_PRIORITY
CREATE_NODEGROUP	CREATE_DB_AT_NODE
DROP_NODEGROUP	KILLDBM
ALTER_NODEGROUP	MIGRATE_SYSTEM_DIRECTORY
CREATE_BUFFERPOOL	DB2REMOT
DROP_BUFFERPOOL	DB2AUD
ALTER_BUFFERPOOL	MERGE_DBM_CONFIG_FILE
CREATE_EVENT_MONITOR	UPDATE_CLI_CONFIGURATION
DROP_EVENT_MONITOR	OPEN_TABLESPACE_QUERY
ENABLE_MULTIPAGE	SINGLE_TABLESPACE_QUERY
MIGRATE_DB_DIR	CLOSE_TABLESPACE_QUERY
DB2TRC	FETCH_TABLESPACE
DB2SET	OPEN_CONTAINER_QUERY
ACTIVATE_DB	FETCH_CONTAINER_QUERY
ADD_NODE	CLOSE_CONTAINER_QUERY
BACKUP_DB	GET_TABLESPACE_STATISTICS
CATALOG_NODE	DESCRIBE_DATABASE
CATALOG_DB	ESTIMATE_SNAPSHOT_SIZE
CATALOG_DCS_DB	READ_ASYNC_LOG_RECORD
CHANGE_DB_COMMENT	PRUNE_RECOVERY_HISTORY
DEACTIVATE_DB	UPDATE_RECOVERY_HISTORY
DROP_NODE_VERIFY	QUIESCE_TABLESPACE
FORCE_APPLICATION	UNLOAD_TABLE
GET_SNAPSHOT	UPDATE_DATABASE_VERSION
LIST_DRDA_INDOUBT_TRANSACTIONS	CREATE_INSTANCE
MIGRATE_DB	DELETE_INSTANCE
RESET_ADMIN_CFG	SET_EVENT_MONITOR
RESET_DB_CFG	GRANT_DBADM
RESET_DBM_CFG	REVOKE_DBADM
RESET_MONITOR	GRANT_DB_AUTHORITIES
RESTORE_DB	REVOKE_DB_AUTHORITIES
	REDIST_NODEGROUP

Related reference:

- "Audit record layout for SYSADMIN events" on page 226

Audit record layout for VALIDATE events

Sample audit record:

```
timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;audit event=CHECK_GROUP_MEMBERSHIP;
  event correlator=2;event status=-1092;
  database=FOO;userid=boss;authid=BOSS;execution id=newton;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  auth type=SERVER;
```

Table 65. Audit Record Layout for VALIDATE Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: VALIDATE
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: GET_GROUPS, GET_USERID, AUTHENTICATE_PASSWORD, VALIDATE_USER, and GET_USERMAPPING_FROM_PLUGIN.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Event Status	INTEGER	Status of audit event, represented by an SQLCODE where Successful event > = 0 Failed event < 0
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Execution ID	VARCHAR(1024)	Execution ID in use at the time of the audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application name in use at the time the audit event occurred.
Authentication Type	VARCHAR (32)	Authentication type at the time of the audit event.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.
Plug-in Name	VARCHAR(32)	The name of the plug-in in use at the time the audit event occurred.

Related concepts:

- "Audit facility record layouts (introduction)" on page 212

Audit record layout for CONTEXT events

Sample audit record:

```
timestamp=1998-06-24-08.42.41.476840;category=CONTEXT;audit event=EXECUTE_IMMEDIATE;
  event correlator=3;
  database=FOO;userid=boss;authid=BOSS;
  application id=*LOCAL.newton.980624124210;application name=testapp;
  package schema=NULLID;package name=SQLC28A1;
  package section=203;text=create table audit(c1 char(10), c2 integer);
```

Table 66. Audit Record Layout for CONTEXT Events

NAME	FORMAT	DESCRIPTION
Timestamp	CHAR(26)	Date and time of the audit event.
Category	CHAR(8)	Category of audit event. Possible values are: CONTEXT
Audit Event	VARCHAR(32)	Specific Audit Event. Possible values include: Those shown in the list following this table.
Event Correlator	INTEGER	Correlation identifier for the operation being audited. Can be used to identify what audit records are associated with a single event.
Database Name	CHAR(8)	Name of the database for which the event was generated. Blank if this was an instance level audit event.
User ID	VARCHAR(1024)	User ID at time of audit event.
Authorization ID	VARCHAR(128)	Authorization ID at time of audit event.
Origin Node Number	SMALLINT	Node number at which the audit event occurred.
Coordinator Node Number	SMALLINT	Node number of the coordinator node.
Application ID	VARCHAR (255)	Application ID in use at the time the audit event occurred.
Application Name	VARCHAR (1024)	Application name in use at the time the audit event occurred.
Package Schema	VARCHAR (128)	Schema of the package in use at the time of the audit event.
Package Name	VARCHAR (128)	Name of package in use at the time the audit event occurred.
Package Section Number	SMALLINT	Section number in package being used at the time the audit event occurred.
Statement Text (statement)	CLOB (2M)	Text of the SQL or XQuery statement, if applicable. Null if no SQL or XQuery statement text is available.
Package Version	VARCHAR (64)	Version of the package in use at the time the audit event occurred.

Related concepts:
- "Audit facility record layouts (introduction)" on page 212

Related reference:
- "List of possible CONTEXT audit events" on page 229

List of possible CONTEXT audit events

The following is the list of possible CONTEXT audit events:

Table 67. CONTEXT Audit Events

CONNECT	SET_APPL_PRIORITY
CONNECT_RESET	RESET_DB_CFG
ATTACH	GET_DB_CFG
DETACH	GET_DFLT_CFG
DARI_START	UPDATE_DBM_CFG
DARI_STOP	SET_MONITOR
BACKUP_DB	GET_SNAPSHOT
RESTORE_DB	ESTIMATE_SNAPSHOT_SIZE
ROLLFORWARD_DB	RESET_MONITOR
OPEN_TABLESPACE_QUERY	OPEN_HISTORY_FILE
FETCH_TABLESPACE	CLOSE_HISTORY_FILE
CLOSE_TABLESPACE_QUERY	FETCH_HISTORY_FILE
OPEN_CONTAINER_QUERY	SET_RUNTIME_DEGREE
CLOSE_CONTAINER_QUERY	UPDATE_AUDIT
FETCH_CONTAINER_QUERY	DBM_CFG_OPERATION
SET_TABLESPACE_CONTAINERS	DISCOVER
GET_TABLESPACE_STATISTIC	OPEN_CURSOR
READ_ASYNC_LOG_RECORD	CLOSE_CURSOR
QUIESCE_TABLESPACE	FETCH_CURSOR
LOAD_TABLE	EXECUTE
UNLOAD_TABLE	EXECUTE_IMMEDIATE
UPDATE_RECOVERY_HISTORY	PREPARE
PRUNE_RECOVERY_HISTORY	DESCRIBE
SINGLE_TABLESPACE_QUERY	BIND
LOAD_MSG_FILE	REBIND
UNQUIESCE_TABLESPACE	RUNSTATS
ENABLE_MULTIPAGE	REORG
DESCRIBE_DATABASE	REDISTRIBUTE
DROP_DATABASE	COMMIT
CREATE_DATABASE	ROLLBACK
ADD_NODE	REQUEST_ROLLBACK
FORCE_APPLICATION	IMPLICIT_REBIND

Related reference:

- "Audit record layout for CONTEXT events" on page 228

Audit facility tips and techniques

In most cases, when working with CHECKING events, the object type field in the audit record is the object being checked to see if the required privilege or authority is held by the user ID attempting to access the object. For example, if a user attempts to ALTER a table by adding a column, then the CHECKING event audit record will indicate the access attempted was "ALTER" and the object type being checked was "TABLE" (note: not the column since it is table privileges that must be checked).

However, when the checking involves verifying if a database authority exists to allow a user ID to CREATE or BIND an object, or to delete an object, then although there is a check against the database, the object type field will specify the object being created, bound, or dropped (rather than the database itself).

When creating an index on a table, the privilege to create an index is required, therefore the CHECKING event audit record will have an access attempt type of "index" rather than "create".

When binding a package that already exists, then an OBJMAINT event audit record is created for the DROP of the package and then another OBJMAINT event audit record is created for the CREATE of the new copy of the package.

Data Definition Language (DDL) may generate OBJMAINT or SECMAINT events that are logged as successful. It is possible however that following the logging of the event, a subsequent error may cause a ROLLBACK to occur. This would leave the object as not created; or the GRANT or REVOKE actions as incomplete. The use of CONTEXT events becomes important in this case. Such CONTEXT event audit records, especially the statement that ends the event, will indicate the nature of the completion of the attempted operation.

When extracting audit records in a delimited ASCII format suitable for loading into a DB2 database relational table, you should be clear regarding the delimiter used within the statement text field. This can be done when extracting the delimited ASCII file and is done using:

```
db2audit extract delasc delimiter <load delimiter>
```

The *load delimiter* can be a single character (such as ") or a four-byte string representing a hexadecimal value (such as “0xff”). Examples of valid commands are:

```
db2audit extract delasc
db2audit extract delasc delimiter !
db2audit extract delasc delimiter 0xff
```

If you have used anything other than the default load delimiter (“"”) as the delimiter when extracting, you should use the MODIFIED BY option on the LOAD command. A partial example of the LOAD command with “0xff” used as the delimiter follows:

```
db2 load from context.del of del modified by chardel0xff replace into ...
```

This will override the default load character string delimiter which is “0xff”.

Related concepts:

- “Audit facility record layouts (introduction)” on page 212

Related reference:

- “Audit facility usage” on page 200

Controlling DB2 database audit facility activities

Procedure:

As part of our discussion on the control of the audit facility activities, we will use a simple scenario: A user, *newton*, runs an application called *testapp* that connects and creates a table. This same application is used in each of the examples discussed below.

We begin by presenting an extreme example: You have determined to audit all successful and unsuccessful audit events, therefore you will configure the audit facility in the following way:

```
db2audit configure scope all status both
```

Note: This creates audit records for every possible auditable event. As a result, many records are written to the audit log and this reduces the performance of your database manager. This extreme case is shown here for demonstration purposes only; there is no recommendation that you configure the audit facility with the command shown above.

After beginning the audit facility with this configuration (using "db2audit start"), and then running the *testapp* application, the following records are generated and placed in the audit log. By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.10.555345;category=CONTEXT;
audit event=CONNECT;event correlator=2;database=FOO;
application id=*LOCAL.newton.980624124210;
application name=testapp;

timestamp=1998-06-24-08.42.10.944374;category=VALIDATE;
audit event=AUTHENTICATION;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;execution id=newton;
application id=*LOCAL.newton.980624124210;application name=testapp;
auth type=SERVER;

timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.561187;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.594620;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.622984;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
object name=FOO;object type=DATABASE;access approval reason=DATABASE;
access attempted=CONNECT;

timestamp=1998-06-24-08.42.11.801554;category=CONTEXT;
audit event=COMMIT;event correlator=2;database=FOO;userid=boss;
authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;

timestamp=1998-06-24-08.42.41.450975;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;object schema=NULLID;
object name=SQLC28A1;object type=PACKAGE;
access approval reason=OBJECT;access attempted=EXECUTE;
```

CREATE TABLE

```
timestamp=1998-06-24-08.42.41.476840;category=CONTEXT;
audit event=EXECUTE_IMMEDIATE;event correlator=3;database=FOO;
userid=boss;authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;package schema=NULLID;package name=SQLC28A1;
package section=203;text=create table audit(c1 char(10), c2 integer);

timestamp=1998-06-24-08.42.41.539692;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object schema=BOSS;object name=AUDIT;object type=TABLE;
access approval reason=DATABASE;access attempted=CREATE;

timestamp=1998-06-24-08.42.41.570876;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object name=BOSS;object type=SCHEMA;access approval reason=DATABASE;
access attempted=CREATE;

timestamp=1998-06-24-08.42.41.957524;category=OBJMAINT;
audit event=CREATE_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object schema=BOSS;object name=AUDIT;object type=TABLE;

timestamp=1998-06-24-08.42.42.018900;category=CONTEXT;
audit event=COMMIT;event correlator=3;database=FOO;userid=boss;
authid=BOSS;application id=*LOCAL.newton.980624124210;
application name=testapp;package schema=NULLID;
package name=SQLC28A1;
```

As you can see, there are a significant number of audit records generated from the audit configuration that requests the auditing of all possible audit events and types.

In most cases, you will configure the audit facility for a more restricted or focused view of the events you wish to audit. For example, you may want to only audit those events that fail. In this case, the audit facility could be configured as follows:

```
db2audit configure scope audit,checking,objmaint,secmaint,sysadmin,
   validate status failure
```

Note: This configuration is the initial audit configuration or the one that occurs when the audit configuration is reset.

After beginning the audit facility with this configuration, and then running the *testapp* application, the following records are generated and placed in the audit log. (And we assume *testapp* has not been run before.) By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.11.527490;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;
```

```
timestamp=1998-06-24-08.42.11.561187;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;

timestamp=1998-06-24-08.42.11.594620;category=VALIDATE;
audit event=CHECK_GROUP_MEMBERSHIP;event correlator=2;
event status=-1092;database=FOO;userid=boss;authid=BOSS;
execution id=newton;application id=*LOCAL.newton.980624124210;
application name=testapp;auth type=SERVER;
```

CREATE TABLE

```
(none)
```

The are far fewer audit records generated from the audit configuration that requests the auditing of all possible audit events (except CONTEXT) but only when the event attempt fails. By changing the audit configuration you can control the type and nature of the audit records that are generated.

The audit facility can allow you to create audit records when those you want to audit have been successfully granted privileges on an object. In this case, you could configure the audit facility as follows:

```
db2audit configure scope checking status success
```

After beginning the audit facility with this configuration, and then running the *testapp* application, the following records are generated and placed in the audit log. (And we assume *testapp* has not been run before.) By extracting the audit records from the log, you will see the following records generated for the two actions carried out by the application:

Action Type of Record Created

CONNECT

```
timestamp=1998-06-24-08.42.11.622984;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;

timestamp=1998-06-24-08.42.41.450975;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=2;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;object schema=NULLID;
object name=SQLC28A1;object type=PACKAGE;
access approval reason=OBJECT;access attempted=EXECUTE;

timestamp=1998-06-24-08.42.41.539692;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object schema=BOSS;object name=AUDIT;object type=TABLE;
access approval reason=DATABASE;access attempted=CREATE;

timestamp=1998-06-24-08.42.41.570876;category=CHECKING;
audit event=CHECKING_OBJECT;event correlator=3;event status=0;
database=FOO;userid=boss;authid=BOSS;
application id=*LOCAL.newton.980624124210;application name=testapp;
package schema=NULLID;package name=SQLC28A1;package section=0;
object name=BOSS;object type=SCHEMA;access approval reason=DATABASE;
access attempted=CREATE;
```

CREATE TABLE
`(none)`

Related concepts:
- "Audit facility record layouts (introduction)" on page 212

Related reference:
- "Audit facility usage" on page 200

Part 6. Setting up the database environment

Chapter 22. Considerations for Creating a Database System

Database directories and files

When you create a database, information about the database including default information is stored in a directory hierarchy. The hierarchical directory structure is created for you at a location that is determined by the information you provide in the CREATE DATABASE command. If you do not specify the location of the directory path or drive when you create the database, the default location is used.

It is recommended that you explicitly state where you would like the database created.

In the directory you specify in the CREATE DATABASE command, a subdirectory that uses the name of the instance is created. This subdirectory ensures that databases created in different instances under the same directory do not use the same path. Below the instance-name subdirectory, a subdirectory named NODE0000 is created. This subdirectory differentiates database partitions in a logically partitioned database environment. Below the node-name directory, a subdirectory named SQL00001 is created. This name of this subdirectory uses the database token and represents the database being created. SQL00001 contains objects associated with the first database created, and subsequent databases are given higher numbers: SQL00002, and so on. These subdirectories differentiate databases created in this instance on the directory that you specified in the CREATE DATABASE command.

The directory structure appears as follows:

```
<your_directory>/<your_instance>/NODE0000/SQL00001/
```

The database directory contains the following files that are created as part of the CREATE DATABASE command.

- The files SQLBP.1 and SQLBP.2 contain buffer pool information. Each file has a duplicate copy to provide a backup.
- The files SQLSPCS.1 and SQLSPCS.2 contain table space information. Each file has a duplicate copy to provide a backup.
- The files SQLSGF.1 and SQLSGF.2 contain storage path information associated with the database's automatic storage. Each file has a duplicate copy to provide a backup.

- The SQLDBCON file contains database configuration information. Do not edit this file. To change configuration parameters, use either the Control Center or the command-line statements UPDATE DATABASE CONFIGURATION and RESET DATABASE CONFIGURATION.
- The DB2RHIST.ASC history file and its backup DB2RHIST.BAK contain history information about backups, restores, loading of tables, reorganization of tables, altering of a table space, and other changes to a database.

  The DB2TSCHNG.HIS file contains a history of table space changes at a log-file level. For each log file, DB2TSCHG.HIS contains information that helps to identify which table spaces are affected by the log file. Table space recovery uses information from this file to determine which log files to process during table space recovery. You can examine the contents of both history files in a text editor.
- The log control files, SQLOGCTL.LFH and SQLOGMIR.LFH, contain information about the active logs.

  Recovery processing uses information from this file to determine how far back in the logs to begin recovery. The SQLOGDIR subdirectory contains the actual log files.

  Note: You should ensure the log subdirectory is mapped to different disks than those used for your data. A disk problem could then be restricted to your data or the logs but not both. This can provide a substantial performance benefit because the log files and database containers do not compete for movement of the same disk heads. To change the location of the log subdirectory, change the *newlogpath* database configuration parameter.
- The SQLINSLK file helps to ensure that a database is used by only one instance of the database manager.

At the same time a database is created, a detailed deadlocks event monitor is also created. The detailed deadlocks event monitor files are stored in the database directory of the catalog node. When the event monitor reaches its maximum number of files to output, it will deactivate and a message is written to the notification log. This prevents the event monitor from consuming too much disk space. Removing output files that are no longer needed will allow the event monitor to activate again on the next database activation.

Additional information for SMS database directories

The SQLT* subdirectories contain the default System Managed Space (SMS) table spaces required for an operational database. Three default table spaces are created:
- SQLT0000.0 subdirectory contains the catalog table space with the system catalog tables.
- SQLT0001.0 subdirectory contains the default temporary table space.
- SQLT0002.0 subdirectory contains the default user data table space.

Each subdirectory or container has a file created in it called SQLTAG.NAM. This file marks the subdirectory as being in use so that subsequent table space creation does not attempt to use these subdirectories.

In addition, a file called SQL*.DAT stores information about each table that the subdirectory or container contains. The asterisk (*) is replaced by a unique set of digits that identifies each table. For each SQL*.DAT file there might be one or more of the following files, depending on the table type, the reorganization status of the table, or whether indexes, LOB, or LONG fields exist for the table:

- SQL*.BKM (contains block allocation information if it is an MDC table)
- SQL*.LF (contains LONG VARCHAR or LONG VARGRAPHIC data)
- SQL*.LB (contains BLOB, CLOB, or DBCLOB data)
- SQL*.XDA (contains XML data)
- SQL*.LBA (contains allocation and free space information about SQL*.LB files)
- SQL*.INX (contains index table data)
- SQL*.IN1 (contains index table data)
- SQL*.DTR (contains temporary data for a reorganization of an SQL*.DAT file)
- SQL*.LFR (contains temporary data for a reorganization of an SQL*.LF file)
- SQL*.RLB (contains temporary data for a reorganization of an SQL*.LB file)
- SQL*.RBA (contains temporary data for a reorganization of an SQL*.LBA file)

Related concepts:
- "Comparison of SMS and DMS table spaces" on page 267
- "Database managed space" on page 265
- "DMS device considerations" in *Administration Guide: Planning*
- "DMS table spaces" in *Administration Guide: Planning*
- "SMS table spaces" in *Administration Guide: Planning*
- "Understanding the recovery history file" in *Data Recovery and High Availability Guide and Reference*

Related reference:
- "CREATE DATABASE " on page 461

Space requirements for database objects

Estimating the size of database objects is an imprecise undertaking. Overhead caused by disk fragmentation, free space, and the use of variable length columns makes size estimation difficult, because there is such a wide range of possibilities for column types and row lengths. After initially estimating your database size, create a test database and populate it with representative data.

From the Control Center, you can access a number of utilities that are designed to assist you in determining the size requirements of various database objects:
- You can select an object and then use the "Estimate Size" utility. This utility can tell you the current size of an existing object, such as a table. You can then change the object, and the utility will calculate new estimated values for the object. The utility will help you approximate storage requirements, taking future growth into account. It provides possible size ranges for the object: both the smallest size, based on current values, and the largest possible size.
- You can determine the relationships between objects by using the "Show Related" window.
- You can select any database object on the instance and request "Generate DDL". This function uses the `db2look` utility to generate data definition statements for the database.

In each of these cases, either the "Show SQL" or the "Show Command" button is available to you. You can save the resulting SQL statements or commands in script files to be used later. All of these utilities have online help to assist you.

Keep these utilities in mind as you plan your physical database.

When estimating the size of a database, the contribution of the following must be considered:

- System Catalog Tables
- User Table Data
- Long Field Data
- Large Object (LOB) Data
- Index Space
- Log File Space
- Temporary Work Space

Space requirements related to the following are not discussed:

- The local database directory file
- The system database directory file
- The file management overhead required by the operating system, including:
  - file block size
  - directory control space

Related concepts:

- "Space requirements for indexes" on page 246
- "Space requirements for large object data" on page 245
- "Space requirements for log files" on page 248
- "Space requirements for long field data" on page 244
- "Space requirements for system catalog tables" on page 242
- "Space requirements for temporary tables" on page 249
- "Space requirements for user table data" on page 243

Related reference:

- "db2look - DB2 statistics and DDL extraction tool command" in *Command Reference*

Space requirements for system catalog tables

System catalog tables are created when a database is created. The system tables grow as database objects and privileges are added to the database. Initially, they use approximately 3.5 MB of disk space.

The amount of space allocated for the catalog tables depends on the type of table space, and the extent size of the table space containing the catalog tables. For example, if a DMS table space with an extent size of 32 is used, the catalog table space is initially allocated 20 MB of space.

Note: For databases with multiple partitions, the catalog tables reside only on the database partition from which the CREATE DATABASE command was issued. Disk space for the catalog tables is only required for that database partition.

Related concepts:

- "Space requirements for database objects" on page 241

• “System catalog tables” on page 297

Space requirements for user table data

By default, table data is stored on 4 KB pages. Each page (regardless of page size) contains 68 bytes of overhead for the database manager. This leaves 4028 bytes to hold user data (or rows), although no row on a 4 KB page can exceed 4005 bytes in length. A row will *not* span multiple pages. You can have a maximum of 500 columns when using a 4 KB page size.

Table data pages *do not* contain the data for columns defined with LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB data types. The rows in a table data page do, however, contain a descriptor for these columns.

Rows are usually inserted into a regular table in first-fit order. The file is searched (using a free space map) for the first available space that is large enough to hold the new row. When a row is updated, it is updated in place, unless there is insufficient space left on the page to contain it. If this is the case, a record is created in the original row location that points to the new location in the table file of the updated row.

If the ALTER TABLE APPEND ON statement is invoked, data is always appended, and information about any free space on the data pages is not kept.

If the table has a clustering index defined on it, DB2 Database for Linux, UNIX, and Windows will attempt to physically cluster the data according to the key order of that clustering index. When a row is inserted into the table, DB2 will first look up its key value in the clustering index. If the key value is found, DB2 attempts to insert the record on the data page pointed to by that key; if the key value is not found, the next higher key value is used, so that the record is inserted on the page containing records having the next higher key value. If there is insufficient space on the “target” page in the table, the free space map is used to search neighboring pages for space. Over time, as space on the data pages is completely used up, records are placed further and further from the “target” page in the table. The table data would then be considered unclustered, and a table reorganization can be used to restore clustered order.

If the table is a multidimensional clustering (MDC) table, DB2 will guarantee that records are always physically clustered along one or more defined dimensions, or clustering indexes. When an MDC table is defined with certain dimensions, a block index is created for each of the dimensions, and a composite block index is created which maps cells (unique combinations of dimension values) to blocks. This composite block index is used to determine to which cell a particular record belongs, and exactly which blocks or extents in the table contains records belonging to that cell. As a result, when inserting records, DB2 searches the composite block index for the list of blocks containing records having the same dimension values, and limits the search for space to those blocks only. If the cell does not yet exist, or if there is insufficient space in the cell’s existing blocks, then another block is assigned to the cell and the record is inserted into it. A free space map is still used within blocks to quickly find available space in the blocks.

The number of 4 KB pages for each user table in the database can be estimated by calculating:

```
ROUND DOWN(4028/(average row size + 10)) = records_per_page
```

and then inserting the result into:

```
(number_of_records/records_per_page) * 1.1 = number_of_pages
```

where the average row size is the sum of the average column sizes, and the factor of ″1.1″ is for overhead.

Note: This formula provides only an estimate. The estimate's accuracy is reduced if the record length varies because of fragmentation and overflow records.

You also have the option to create buffer pools or table spaces that have an 8 KB, 16 KB, or 32 KB page size. All tables created within a table space of a particular size have a matching page size. A single table or index object can be as large as 512 GB, assuming a 32 KB page size. You can have a maximum of 1012 columns when using an 8 KB, 16 KB, or 32 KB page size. The maximum number of columns is 500 for a 4 KB page size. Maximum row lengths also vary, depending on page size:

- When the page size is 4 KB, the row length can be up to 4005 bytes.
- When the page size is 8 KB, the row length can be up to 8101 bytes.
- When the page size is 16 KB, the row length can be up to 16 293 bytes.
- When the page size is 32 KB, the row length can be up to 32 677 bytes.

A larger page size facilitates a reduction in the number of levels in any index. If you are working with OLTP (online transaction processing) applications, that perform random row reads and writes, a smaller page size is better, because it wastes less buffer space with undesired rows. If you are working with DSS (decision support system) applications, which access large numbers of consecutive rows at a time, a larger page size is better because it reduces the number of I/O requests required to read a specific number of rows. An exception occurs when the row size is smaller than the page size divided by 255. In such a case, there is wasted space on each page. (There is a maximum of only 255 rows per page.) To reduce this wasted space, a smaller page size may be appropriate.

You cannot restore a backup to a different page size.

You cannot import IXF data files that represent more than 755 columns.

Declared temporary tables can be created only in their own "user temporary" table space type. There is no default user temporary table space. Temporary tables cannot have LONG data. The tables are dropped implicitly when an application disconnects from the database, and estimates of the space requirements for their tables should take this into account.

Related concepts:

- "Space requirements for database objects" on page 241

Space requirements for long field data

Long field data is stored in a separate table object that is structured differently than the storage space for other data types.

Data is stored in 32 KB areas that are broken up into segments whose sizes are ″powers of two″ times 512 bytes. (Hence these segments can be 512 bytes, 1024 bytes, 2048 bytes, and so on, up to 32 768 bytes.)

Long field data types (LONG VARCHAR or LONG VARGRAPHIC) are stored in a way that enables free space to be reclaimed easily. Allocation and free space information is stored in 4 KB allocation pages, which appear infrequently throughout the object.

The amount of unused space in the object depends on the size of the long field data, and whether this size is relatively constant across all occurrences of the data. For data entries larger than 255 bytes, this unused space can be up to 50 percent of the size of the long field data.

If character data is less than the page size, and it fits into the record along with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of LONG VARCHAR or LONG VARGRAPHIC.

Related concepts:
- "Space requirements for database objects" on page 241

Space requirements for large object data

Large Object (LOB) data is stored in two separate table objects that are structured differently than the storage space for other data types.

To estimate the space required by LOB data, you need to consider the two table objects used to store data defined with these data types:
- **LOB Data Objects**

  Data is stored in 64 MB areas that are broken up into segments whose sizes are ″powers of two″ times 1024 bytes. (Hence these segments can be 1024 bytes, 2048 bytes, 4096 bytes, and so on, up to 64 MB.)

  To reduce the amount of disk space used by LOB data, you can specify the COMPACT option on the *lob-options* clause of the CREATE TABLE and the ALTER TABLE statements. The COMPACT option minimizes the amount of disk space required by allowing the LOB data to be split into smaller segments. This process does not involve data compression, but simply uses the minimum amount of space, to the nearest 1 KB boundary. Using the COMPACT option may result in reduced performance when appending to LOB values.

  The amount of free space contained in LOB data objects is influenced by the amount of update and delete activity, as well as the size of the LOB values being inserted.
- **LOB Allocation Objects**

  Allocation and free space information is stored in 4 KB allocation pages that are separated from the actual data. The number of these 4 KB pages is dependent on the amount of data, including unused space, allocated for the large object data. The overhead is calculated as follows: one 4 KB page for every 64 GB, plus one 4 KB page for every 8 MB.

If character data is less than the page size, and it fits into the record along with the rest of the data, the CHAR, GRAPHIC, VARCHAR, or VARGRAPHIC data types should be used instead of BLOB, CLOB, or DBCLOB.

Related concepts:
- "Space requirements for database objects" on page 241

Related reference:

- “Large objects (LOBs)” in *SQL Reference, Volume 1*

Space requirements for indexes

For each index, the space needed can be estimated as:

```
(average index key size + 9) * number of rows * 2
```

where:

- The “average index key size” is the byte count of each column in the index key. (When estimating the average column size for VARCHAR and VARGRAPHIC columns, use an average of the current data size, plus two bytes. Do not use the maximum declared size.)
- The factor of “2” is for overhead, such as non-leaf pages and free space.

Notes:

1. For every column that allows NULLs, add one extra byte for the null indicator.
2. For block indexes created internally for multidimensional clustering (MDC) tables, the “number of rows” would be replaced by the “number of blocks”.

Indexes created before Version 8 (type-1 indexes) are different from those created at version 8 (type-2 indexes) and following. To find out what type of index exists for a table, use the INSPECT command. To convert type-1 indexes to type-2 indexes, use the REORG INDEXES commmand.

When using the REORG INDEXES command, ensure that you have sufficient free space in the table space where the indexes are stored. The amount of free space should be equal to the current size of the index. Additional space may be required if you choose to reorganize the indexes with the ALLOW WRITE ACCESS option. The additional space is for the logs of the activity affecting the indexes during the reorganization of the indexes.

Temporary space is required when creating the index. The maximum amount of temporary space required during index creation can be estimated as:

```
(average index key size + 9) * number of rows * 3.2
```

where the factor of “3.2” is for index overhead, and space required for sorting during index creation.

Note: In the case of non-unique indexes, only five bytes are required to store duplicate key entries. The estimate shown above assumes no duplicates. The space required to store an index may be over-estimated by the formula shown above.

The following two formulas can be used to estimate the number of keys per leaf page (the second provides a more accurate estimate). The accuracy of these estimates depends largely on how well the averages reflect the actual data.

Note: For SMS table spaces, the minimum required space for leaf pages is 12 KB. For DMS table spaces, the minimum is an extent.

- A rough estimate of the average number of keys per leaf page is:

  ```
  (.9 * (U - (M*2))) * (D + 1)
  ----------------------------
       K + 7 + (5 * D)
  ```

where:

– U, the usable space on a page, is approximately equal to the page size minus 100. For a page size of 4096, U is 3996.
– M = U / (9 + *minimumKeySize*)
– D = average number of duplicates per key value
– K = *averageKeySize*

Remember that *minimumKeySize* and *averageKeysize* must have an extra byte for each nullable key part, and an extra two bytes for the length of each variable length key part.

If there are include columns, they should be accounted for in *minimumKeySize* and *averageKeySize*.

The `.9` can be replaced by any (100 - pctfree)/100 value, if a percent free value other than the default value of ten percent is specified during index creation.

- A more accurate estimate of the average number of keys per leaf page is:

```
L = number of leaf pages = X / (avg number of keys on leaf page)
```

where X is the total number of rows in the table.

You can estimate the original size of an index as:

```
(L + 2L/(average number of keys on leaf page)) * pagesize
```

For DMS table spaces, add the sizes of all indexes on a table and round up to a multiple of the extent size for the table space on which the index resides.

You should provide additional space for index growth due to INSERT/UPDATE activity, from which page splits may result.

Use the following calculation to obtain a more accurate estimate of the original index size, as well as an estimate of the number of levels in the index. (This may be of particular interest if include columns are being used in the index definition.) The average number of keys per non-leaf page is roughly:

```
(.9 * (U - (M*2))) * (D + 1)
----------------------------
      K + 13 + (9 * D)
```

where:

– U, the usable space on a page, is approximately equal to the page size minus 100. For a page size of 4096, U is 3996.
– D is the average number of duplicates per key value on non-leaf pages (this will be much smaller than on leaf pages, and you may want to simplify the calculation by setting the value to 0).
– M = U / (9 + *minimumKeySize* for non-leaf pages)
– K = *averageKeySize* for non-leaf pages

The *minimumKeySize* and the *averageKeySize* for non-leaf pages will be the same as for leaf pages, except when there are include columns. Include columns are not stored on non-leaf pages.

You should not replace `.9` with `(100 - pctfree)/100`, unless this value is greater than `.9`, because a maximum of 10 percent free space will be left on non-leaf pages during index creation.

The number of non-leaf pages can be estimated as follows:

```
if L > 1 then {P++; Z++}
While (Y > 1)
{
      P = P + Y
      Y = Y / N
     Z++
   }
```

where:
- P is the number of pages (0 initially).
- L is the number of leaf pages.
- N is the number of keys for each non-leaf page.
- Y = L / N
- Z is the number of levels in the index tree (1 initially).

Total number of pages is:

```
T = (L + P + 2) * 1.0002
```

The additional 0.02 percent is for overhead, including space map pages.

The amount of space required to create the index is estimated as:

```
T * pagesize
```

Related concepts:
- "Indexes" in *SQL Reference, Volume 1*
- "Index cleanup and maintenance" in *Performance Guide*
- "Space requirements for database objects" on page 241

Space requirements for log files

You will require 32 KB of space for log control files.

You will also need at least enough space for your active log configuration, which you can calculate as

```
(logprimary + logsecond) * (logfilsiz + 2 ) * 4096
```

where:
- *logprimary* is the number of primary log files, defined in the database configuration file
- *logsecond* is the number of secondary log files, defined in the database configuration file; in this calculation, *logsecond* cannot be set to *-1*. (When *logsecond* is set to -1, you are requesting an infinite active log space.)
- *logfilsiz* is the number of pages in each log file, defined in the database configuration file
- 2 is the number of header pages required for each log file
- 4096 is the number of bytes in one page.

If the database is enabled for circular logging, the result of this formula will provide a sufficient amount of disk space.

If the database is enabled for roll-forward recovery, special log space requirements should be taken into consideration:
- With the *logretain* configuration parameter enabled, the log files will be archived in the log path directory. The online disk space will eventually fill up, unless you move the log files to a different location.

- With the *userexit* configuration parameter enabled, a user exit program moves the archived log files to a different location. Extra log space is still required to allow for:
  – Online archived logs that are waiting to be moved by the user exit program
  – New log files being formatted for future use

If the database is enabled for infinite logging (that is, you set *logsecond* to *-1*), the *logarchmeth1* configuration parameter must be set to a value other than OFF for LOGRETAIN to enable archive logging. DB2 Database for Linux, UNIX, and Windows will keep at least the number of active log files specified by *logprimary* in the log path, so you should not use the value of -1 for *logsecond* in the above formula. Ensure that you provide extra disk space to allow for the delay caused by archiving log files.

If you are mirroring the log path, you will need to double the estimated log file space requirements.

Related concepts:
- "Space requirements for database objects" on page 241
- "Log mirroring" in *Data Recovery and High Availability Guide and Reference*
- "Understanding recovery logs" on page 1534

Related reference:
- "mirrorlogpath - Mirror log path configuration parameter" in *Performance Guide*
- "logprimary - Number of primary log files configuration parameter" in *Performance Guide*
- "logsecond - Number of secondary log files configuration parameter" in *Performance Guide*
- "logfilsiz - Size of log files configuration parameter" in *Performance Guide*

Space requirements for temporary tables

Some SQL statements require temporary tables for processing (such as a work file for sorting operations that cannot be done in memory). These temporary tables require disk space; the amount of space required is dependent upon the size, number, and nature of the queries, and the size of returned tables. Your work environment is unique which makes the determination of your space requirements for temporary tables difficult to estimate. For example, more space may appear to be allocated for system temporary table spaces than is actually in use due to the longer life of various system temporary tables. This could occur when DB2_SMS_TRUNC_TMPTABLE_THRESH is used.

You can use the database system monitor and the query table space APIs to track the amount of work space being used during the normal course of operations.

You can use the DB2_OPT_MAX_TEMP_SIZE registry variable to limit the amount of temporary table space used by queries.

Related concepts:
- "Space requirements for database objects" on page 241

Related reference:
- "sqlbmtsq - Get the query data for all table spaces" on page 783

- “Query compiler variables” in *Performance Guide*

Database partition groups

A database partition group is a set of one or more database partitions defined as belongin to a database. When you want to create tables for the database, you first create the database partition group where the table spaces will be stored, then you create the table space where the tables will be stored.

You can define named subsets of one or more database partitions in a database. Each subset you define is known as a *database partition group*. Each subset that contains more than one database partition is known as a *multipartition database partition group*. Multipartition database partition groups can only be defined with database partitions that belong to the same instance. A database partition group can contain as few as one database partition, or span all of the database partitions in the database.

Figure 19 on page 51 shows an example of a database with five database partitions in which:

- A database partition group spans all but one of the database partitions (Database Partition Group 1).
- A database partition group contains one database partition (Database Partition Group 2).
- A database partition group contains two database partitions. (Database Partition Group 3).
- The database partition within Database Partition Group 2 is shared (and overlaps) with Database Partition Group 1.
- There is a single database partition within Database Partition Group 3 that is shared (and overlaps) with Database Partition Group 1.

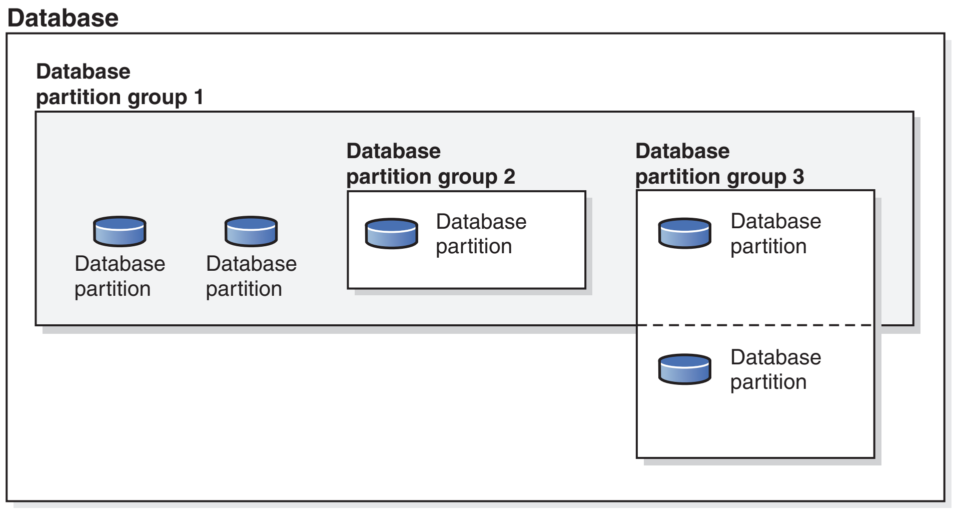


Figure 19. Database partition groups in a database

You create a new database partition group using the CREATE DATABASE PARTITION GROUP statement. You can modify it using the ALTER DATABASE PARTITION GROUP statement. Data is divided across all the database partitions in a database partition group, and you can add or drop one or more database partitions from a database partition group. If you are using a multipartition database partition group, you must look at several database partition group design considerations.

Each database partition that is part of the database system configuration must already be defined in a *database partition configuration file* called `db2nodes.cfg`. A database partition group can contain as few as one database partition, or as many as the entire set of database partitions defined for the database system.

When a database partition group is created or modified, a *distribution map* is associated with it. A distribution map, in conjunction with a *distribution key* and a hashing algorithm, is used by the database manager to determine which database partition in the database partition group will store a given row of data.

In a non-partitioned database, no distribution key or distribution map is required. A database partition is a part of the database, complete with user data, indexes, configuration files, and transaction logs. Default database partition groups that were created when the database was created are used by the database manager. IBMCATGROUP is the default database partition group for the table space containing the system catalogs. IBMTEMPGROUP is the default database partition group for system temporary table spaces. IBMDEFAULTGROUP is the default database partition group for the table spaces containing the user defined tables that you may choose to put there. A user temporary table space for a declared temporary table can be created in IBMDEFAULTGROUP or any user-created database partition group but not in IBMTEMPGROUP.

When working with database partition groups you can:
- Create a database partition group.
- Change the comment associated with a database partition group.
- Add database partitions to a database partition group.
- Drop database partitions from a database partition group.
- Redistribute table data within a database partition group.

Related concepts:
- “Database partition group design” on page 251
- “Distribution keys” on page 254
- “Distribution maps” on page 252

Related reference:
- “ALTER DATABASE PARTITION GROUP ” on page 845
- “CREATE DATABASE PARTITION GROUP ” on page 918

Database partition group design

There are no database partition group design considerations if you are using a single-partition database.

The DB2 Design Advisor is a tool that can be used to recommend database partition groups. The DB2 Design Advisor can be accessed from the Control Center and using db2advis from the command line processor.

If you are using a multiple partition database partition group, consider the following design points:

- In a multipartition database partition group, you can only create a unique index if it is a superset of the distribution key.
- Depending on the number of database partitions in the database, you may have one or more single-partition database partition groups, and one or more multipartition database partition groups present.
- Each database partition must be assigned a unique number. The same database partition may be found in one or more database partition groups.
- To ensure fast recovery of the database partition containing system catalog tables, avoid placing user tables on the same database partition. This is accomplished by placing user tables in database partition groups that do not include the database partition in the IBMCATGROUP database partition group.

You should place small tables in single-partition database partition groups, except when you want to take advantage of *collocation* with a larger table. Collocation is the placement of rows from different tables that contain related data in the same database partition. Collocated tables allow DB2 Database for Linux, UNIX, and Windows to utilize more efficient join strategies. Collocated tables can reside in a single-partition database partition group. Tables are considered collocated if they reside in a multipartition database partition group, have the same number of columns in the distribution key, and if the data types of the corresponding columns are compatible. Rows in collocated tables with the same distribution key value are placed on the same database partition. Tables can be in separate table spaces in the same database partition group, and still be considered collocated.

You should avoid extending medium-sized tables across too many database partitions. For example, a 100 MB table may perform better on a 16-partition database partition group than on a 32-partition database partition group.

You can use database partition groups to separate online transaction processing (OLTP) tables from decision support (DSS) tables, to ensure that the performance of OLTP transactions is not adversely affected.

Related concepts:
- "Database partition groups," on page 250
- "Database partition compatibility" on page 256
- "Distribution keys" on page 254
- "Distribution maps" on page 252
- "Replicated materialized query tables" on page 257
- "Table collocation" on page 255

Related reference:
- "db2advis - DB2 design advisor command" in *Command Reference*

Distribution maps

In a partitioned database environment, the database manager must know where to find the data it needs. The database manager uses a map, called a *distribution map*, to find the data.

A distribution map is an internally generated array containing either 4 096 entries for multiple-partition database partition groups, or a single entry for single-partition database partition groups. For a single-partition database partition group, the distribution map has only one entry containing the number of the database partition where all the rows of a database table are stored. For multiple-partition database partition groups, the numbers of the database partition group are specified in a way such that each database partition is used one after the other to ensure an even distribution across the entire map. Just as a city map is organized into sections using a grid, the database manager uses a *distribution key* to determine the location (the database partition) where the data is stored.

For example, assume that you have a database created on four database partitions (numbered 0–3). The distribution map for the IBMDEFAULTGROUP database partition group of this database would be:

```
0 1 2 3 0 1 2 ...
```

If a database partition group had been created in the database using database partitions 1 and 2, the distribution map for that database partition group would be:

```
1 2 1 2 1 2 1 ...
```

If the distribution key for a table to be loaded into the database is an integer with possible values between 1 and 500 000, the distribution key is hashed to a number between 0 and 4 095. That number is used as an index into the distribution map to select the database partition for that row.

Figure 23 on page 262 shows how the row with the distribution key value (c1, c2, c3) is mapped to number 2, which, in turn, references database partition n5.

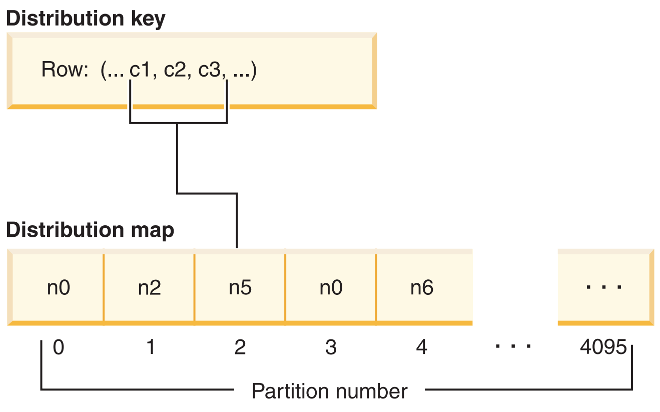


Figure 20. Data distribution using a distribution map

A distribution map is a flexible way of controlling where data is stored in a multi-partition database. If you need to change the data distribution across the database partitions in your database, you can use the data redistribution utility. This utility allows you to rebalance or introduce skew into the data distribution.

You can use the `sqlugtpi API - Get table distribution information`) to obtain a copy of a distribution map that you can view.

Related concepts:

- "Database partition group design" on page 251
- "Database partition groups," on page 250
- "Distribution keys" on page 254

Related reference:

- "sqlugtpi - Get table distribution information" on page 826

Distribution keys

A *distribution key* is a column (or group of columns) that is used to determine the database partition in which a particular row of data is stored. A distribution key is defined on a table using the CREATE TABLE statement. If a distribution key is not defined for a table in a table space that is divided across more than one database partition in a database partition group, one is created by default from the first column of the primary key. If no primary key is specified, the default distribution key is the first non-long field column defined on that table. (*Long* includes all long data types and all large object (LOB) data types). If you are creating a table in a table space associated with a single-partition database partition group, and you want to have a distribution key, you must define the distribution key explicitly. One is not created by default.

If no columns satisfy the requirement for a default distribution key, the table is created without one. Tables without a distribution key are only allowed in single-partition database partition groups. You can add or drop distribution keys later, using the ALTER TABLE statement. Altering the distribution key can only be done to a table whose table space is associated with a single-partition database partition group.

Choosing a good distribution key is important. You should take into consideration:
- How tables are to be accessed
- The nature of the query workload
- The join strategies employed by the database system.

If collocation is not a major consideration, a good distribution key for a table is one that spreads the data evenly across all database partitions in the database partition group. The distribution key for each table in a table space that is associated with a database partition group determines if the tables are collocated. Tables are considered collocated when:
- The tables are placed in table spaces that are in the same database partition group
- The distribution keys in each table have the same number of columns
- The data types of the corresponding columns are partition-compatible.

These characteristics ensure that rows of collocated tables with the same distribution key values are located on the same database partition.

An inappropriate distribution key can cause uneven data distribution. Columns with unevenly distributed data, and columns with a small number of distinct values should not be chosen as distribution keys. The number of distinct values must be great enough to ensure an even distribution of rows across all database partitions in the database partition group. The cost of applying the distribution algorithm is proportional to the size of the distribution key. The distribution key cannot be more than 16 columns, but fewer columns result in better performance. Unnecessary columns should not be included in the distribution key.

The following points should be considered when defining distribution keys:
- Creation of a multiple-partition table that contains only long data types (LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, or DBCLOB) is not supported.
- The distribution key definition cannot be altered.
- The distribution key should include the most frequently joined columns.

- The distribution key should be made up of columns that often participate in a GROUP BY clause.
- Any unique key or primary key must contain all of the distribution key columns.
- In an online transaction processing (OLTP) environment, all columns in the distribution key should participate in the transaction by using equal (=) predicates with constants or host variables. For example, assume you have an employee number, *emp_no*, that is often used in transactions such as:

  ```
  UPDATE emp_table SET ... WHERE
  emp_no = host-variable
  ```

  In this case, the EMP_NO column would make a good single column distribution key for EMP_TABLE.

Database partitioning is the method by which the placement of each row in the table is determined. The method works as follows:

1. A hashing algorithm is applied to the value of the distribution key, and generates a number between zero (0) and 4095.
2. The distribution map is created when a database partition group is created. Each of the numbers is sequentially repeated in a round-robin fashion to fill the distribution map.
3. The number is used as an index into the distribution map. The number at that location in the distribution map is the number of the database partition where the row is stored.

Related concepts:

- "Database partition group design" on page 251
- "Database partition groups," on page 250
- "Distribution maps" on page 252
- "The Design Advisor" in *Performance Guide*

Related reference:

- "ALTER TABLE " on page 854

Table collocation

You may discover that two or more tables frequently contribute data in response to certain queries. In this case, you will want related data from such tables to be located as close together as possible. In an environment where the database is physically divided among two or more database partitions, there must be a way to keep the related pieces of the divided tables as close together as possible. The ability to do this is called *table collocation*.

Tables are collocated when they are stored in the same database partition group, and when their distribution keys are compatible. Placing both tables in the same database partition group ensures a common distribution map. The tables may be in different table spaces, but the table spaces must be associated with the same database partition group. The data types of the corresponding columns in each distribution key must be *partition-compatible*.

DB2 Database for Linux, UNIX, and Windows can recognize, when accessing more than one table for a join or a subquery, that the data to be joined is located at the same database partition. When this happens, DB2 can perform the join or subquery at the database partition where the data is stored, instead of having to move data between database partitions. This ability has significant performance advantages.

Related concepts:
- "Database partition group design" on page 251
- "Database partition groups," on page 250
- "Database partition compatibility" on page 256
- "Distribution keys" on page 254

Database partition compatibility

The base data types of corresponding columns of distribution keys are compared and can be declared *partition-compatible*. Partition-compatible data types have the property that two variables, one of each type, with the same value, are mapped to the same number by the same partitioning algorithm.

Partition-compatibility has the following characteristics:
- A base data type is compatible with another of the same base data type.
- Internal formats are used for DATE, TIME, and TIMESTAMP data types. They are not compatible with each other, and none are compatible with CHAR.
- Partition-compatibility is not affected by columns with NOT NULL or FOR BIT DATA definitions.
- NULL values of compatible data types are treated identically; those of non-compatible data types may not be.
- Base data types of a user-defined type are used to analyze partition-compatibility.
- Decimals of the same value in the distribution key are treated identically, even if their scale and precision differ.
- Trailing blanks in character strings (CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC) are ignored by the hashing algorithm.
- BIGINT, SMALLINT, and INTEGER are compatible data types.
- REAL and FLOAT are compatible data types.
- CHAR and VARCHAR of different lengths are compatible data types.
- GRAPHIC and VARGRAPHIC are compatible data types.
- Partition-compatibility does not apply to LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, and BLOB data types, because they are not supported as distribution keys.

Related concepts:
- "Database partition group design" on page 251
- "Database partition groups," on page 250
- "Distribution keys" on page 254

Replicated materialized query tables

A *materialized query table* is a table that is defined by a query that is also used to determine the data in the table. Materialized query tables can be used to improve the performance of queries. If DB2 Database for Linux, UNIX, and Windows determines that a portion of a query could be resolved using a materialized query table, the query may be rewritten by the database manager to use the materialized query table.

In a partitioned database environment, you can replicate materialized query tables and use them to improve query performance. A *replicated materialized query table* is based on a table that may have been created in a single-partition database partition group, but that you want replicated across all of the database partitions in another database partition group. To create the replicated materialized query table, invoke the CREATE TABLE statement with the REPLICATED keyword.

By using replicated materialized query tables, you can obtain collocation between tables that are not typically collocated. Replicated materialized query tables are particularly useful for joins in which you have a large fact table and small dimension tables. To minimize the extra storage required, as well as the impact of having to update every replica, tables that are to be replicated should be small and updated infrequently.

Note: You should also consider replicating larger tables that are updated infrequently: the one-time cost of replication is offset by the performance benefits that can be obtained through collocation.

By specifying a suitable predicate in the subselect clause used to define the replicated table, you can replicate selected columns, selected rows, or both.

Related concepts:
- "Database partition group design" on page 251
- "The Design Advisor" in *Performance Guide*

Related tasks:
- "Creating a materialized query table" in *Administration Guide: Implementation*

Related reference:
- "CREATE TABLE " on page 956

Table space design

A table space is a storage structure containing tables, indexes, large objects, and long data. Table spaces reside in database partition groups. They allow you to assign the location of database and table data directly onto containers. (A container can be a directory name, a device name, or a file name.) This can provide improved performance and more flexible configuration.

Since table spaces reside in database partition groups, the table space selected to hold a table defines how the data for that table is distributed across the database partitions in a database partition group. A single table space can span several containers. It is possible for multiple containers (from one or more table spaces) to be created on the same physical disk (or drive). For improved performance, each container should use a different disk. Figure 21 on page 258 illustrates the relationship between tables and table spaces within a database, and the containers associated with that database.

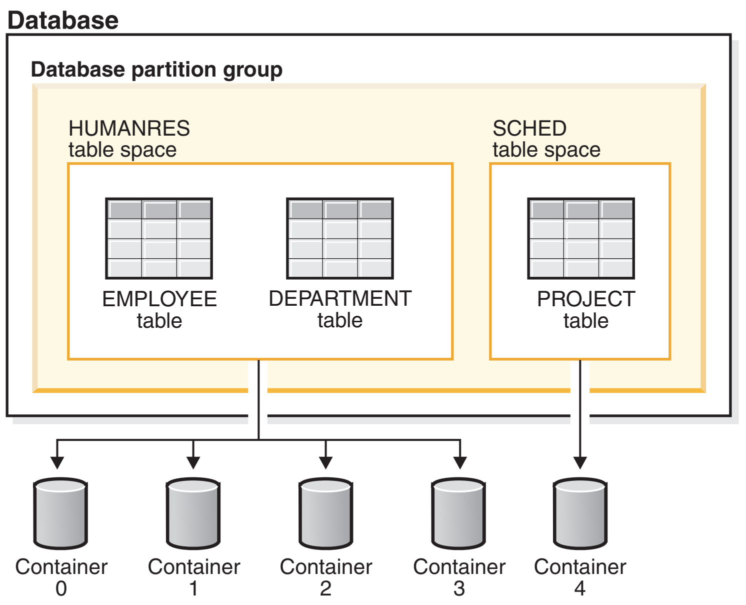


Figure 21. Table spaces and tables in a database

The EMPLOYEE and DEPARTMENT tables are in the HUMANRES table space, which spans containers 0, 1, 2 and 3. The PROJECT table is in the SCHED table space in container 4. This example shows each container existing on a separate disk.

The database manager attempts to balance the data load across containers. As a result, all containers are used to store data. The number of pages that the database manager writes to a container before using a different container is called the *extent size*. The database manager does not always start storing table data in the first container.

Figure 22 on page 259 shows the HUMANRES table space with an extent size of two 4 KB pages, and four containers, each with a small number of allocated extents. The DEPARTMENT and EMPLOYEE tables both have seven pages, and span all four containers.

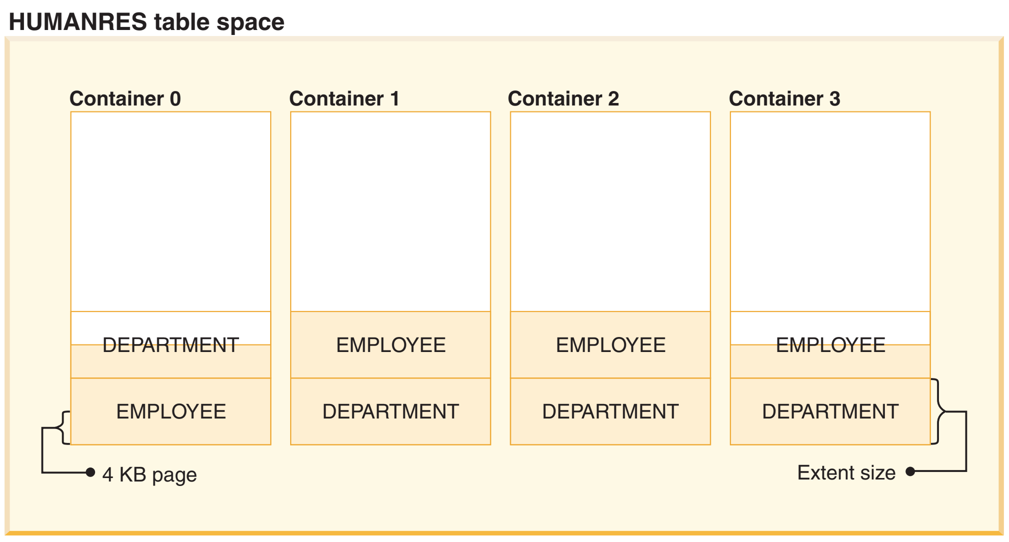


Figure 22. Containers and extents in a table space

A database must contain at least three table spaces:

- One *catalog table space*, which contains all of the system catalog tables for the database. This table space is called SYSCATSPACE, and it cannot be dropped. IBMCATGROUP is the default database partition group for this table space.
- One or more *user table spaces*, which contain all user defined tables. By default, one table space, USERSPACE1, is created. IBMDEFAULTGROUP is the default database partition group for this table space.

  You should specify a table space name when you create a table, or the results may not be what you intend.

  A table's page size is determined either by row size, or the number of columns. The maximum allowable length for a row is dependent upon the page size of the table space in which the table is created. Possible values for page size are 4 KB, 8 KB, 16 KB, and 32 KB. Before Version 9.1, the default page size was 4 KB. In Version 9.1 and following, the default page size may be one of the other supported values. The default page size is declared when creating a new database. Once the default page size has been declared, you are still free to create a table space with one page size for the base table, and a different table space with a different page size for long or LOB data. (Recall that SMS does not support tables that span table spaces, but that DMS does.) If the number of columns or the row size exceeds the limits for a table space's page size, an error is returned (SQLSTATE 42997).
- One or more *temporary table spaces*, which contain temporary tables. Temporary table spaces can be *system temporary table spaces* or *user temporary table spaces*.

  System temporary table spaces hold temporary data required by the database manager while performing operations such as sorts or joins. These types of operations require extra space to process the results set. A database must have at least one system temporary table space; by default, one system temporary table space called TEMPSPACE1 is created at database creation. IBMTEMPGROUP is the default database partition group for this table space.

  User temporary table spaces hold temporary data from tables created with a DECLARE GLOBAL TEMPORARY TABLE statement. To allow the definition of declared temporary tables, at least one user temporary table space should be created with the appropriate USE privileges. USE privileges are granted using the GRANT statement. A user temporary table spaces is *not* created by default at database creation.

If a database uses more than one temporary table space and a new temporary object is needed, the optimizer will choose an appropriate page size for this object. That object will then be allocated to the temporary table space with the corresponding page size. If there is more than one temporary table space with that page size, then the table space will be chosen in a round-robin fashion. In most circumstances, it is not recommended to have more than one temporary table space of any one page size.

If queries are running against tables in table spaces that are defined with page sizes larger than the default, some of them may fail. This will occur if there are no temporary table spaces defined with a larger page size. You may need to create a temporary table space with a larger page size (if the default was 4 KB, then you would need to create a temporary table space with a page size of 8 KB, 16 KB, or 32 KB). Any DML (Data Manipulation Language) statement could fail unless there exists a temporary table space with the same page size as the largest page size in the user table space.

You should define a single SMS temporary table space with a page size equal to the page size used in the majority of your user table spaces. This should be adequate for typical environments and workloads.

In a partitioned database environment, the catalog node will contain all three default table spaces, and the other database partitions will each contain only TEMPSPACE1 and USERSPACE1.

There are two types of table space, both of which can be used in a single database:

- System managed space, in which the operating system's file manager controls the storage space.
- Database managed space, in which the database manager controls the storage space.

Related concepts:

- "Catalog table space design" on page 272
- "Comparison of SMS and DMS table spaces" on page 267
- "Database managed space" on page 265
- "Extent size" in *Administration Guide: Planning*
- "Relationship between table spaces and buffer pools" on page 269
- "Relationship between table spaces and database partition groups" on page 269
- "System managed space" on page 262
- "SYSTOOLSPACE and SYSTOOLSTMPSPACE table spaces" in *Administration Guide: Planning*
- "Temporary table space design" on page 270
- "Workload considerations in table space design" in *Administration Guide: Planning*
- "Table space disk I/O" in *Administration Guide: Planning*
- "Table spaces and other storage structures" on page 261

Related tasks:

- "Optimizing table space performance when data is on RAID devices" in *Administration Guide: Planning*
- "Creating a table space" on page 299

Related reference:

- “CREATE TABLE ” on page 956
- “CREATE TABLESPACE ” on page 1021

Table spaces and other storage structures

Storage structures contain database objects. The basic storage structure is the *table space*; it contains tables, indexes, large objects, and data defined with a LONG data type. There are two types of table spaces:

Database managed space (DMS)
A table space that is managed by the database manager.

System managed space (SMS)
A table space that is managed by the operating system.

All table spaces consist of containers. A *container* describes where objects are stored. A subdirectory in a file system is an example of a container.

When data is read from table space containers, it is placed in an area of memory called a buffer pool. A *buffer pool* is associated with a specific table space, thereby allowing control over which data will share the same memory areas for data buffering.

In a partitioned database, data is spread across different database partitions. Exactly which database partitions are included is determined by the database partition group that is assigned to the table space. A *database partition group* is a group of one or more database partitions that are defined as part of the database. A table space includes one or more containers for each partition in the database partition group. A *distribution map*, associated with each database partition group, is used by the database manager to determine on which database partition a given row of data is to be stored. The distribution map is an array of 4096 database partition numbers. The distribution map index produced by the partitioning function for each row in a table is used as an index into the distribution map to determine the database partition on which a row is to be stored. As an example, the following figure shows how a row with distribution key value (c1, c2, c3) is mapped to distribution map index 2 which, in turn, references database partition p5.

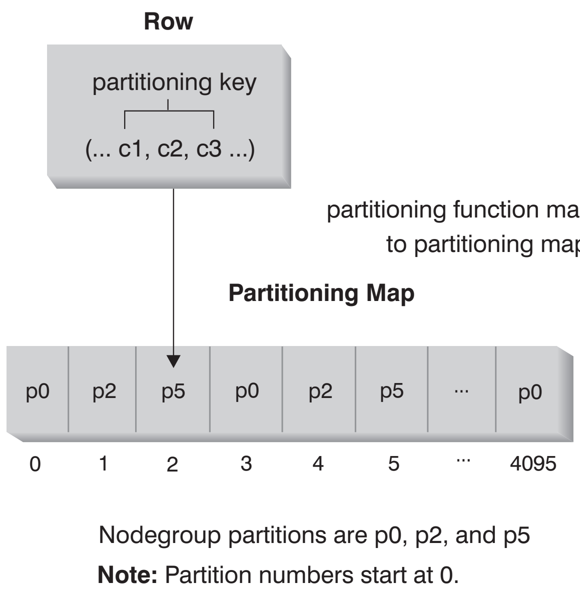

Row
partitioning key
(... c1, c2, c3 ...)
partitioning function maps (c1, c2, c3)
to partitioning map index 2
Partitioning Map
p0 p2 p5 p0 p2 p5 ... p0
0 1 2 3 4 5 ... 4095
Nodegroup partitions are p0, p2, and p5
Note: Partition numbers start at 0.

Figure 23. Data Distribution

The distribution map can be changed, allowing the data distribution to be changed without modifying the distribution key or the actual data. The new distribution map is specified as part of the REDISTRIBUTE DATABASE PARTITION GROUP command or the sqludrdt application programming interface (API), which use it to redistribute the tables in the database partition group.

The DB2 Data Links Manager product provides functionality that supports additional storage capabilities. A normal user table can include columns (defined with the DATALINK data type) that register links to data stored in external files. DATALINK values point to data files that are stored on an external file server.

Related concepts:

- Chapter 9, "Database partitioning across multiple database partitions," on page 49

Related reference:

- "CREATE BUFFERPOOL statement" in *SQL Reference, Volume 2*
- "CREATE DATABASE PARTITION GROUP " on page 918
- "CREATE TABLESPACE " on page 1021

System managed space

In an SMS (System Managed Space) table space, the operating system's file system manager allocates and manages the space where the table is stored. The storage model typically consists of many files, representing table objects, stored in the file system space. The user decides on the location of the files, DB2 Database for Linux, UNIX, and Windows controls their names, and the file system is responsible for managing them. By controlling the amount of data written to each file, the database manager distributes the data evenly across the table space containers.

Each table has at least one SMS physical file associated with it.

The data in the table spaces is striped by extent across all the containers in the system. An *extent* is a group of consecutive pages defined to the database. The file extension denotes the type of the data stored in the file. To distribute the data evenly across all containers in the table space, the starting extents for tables are placed in round-robin fashion across all containers. Such distribution of extents is particularly important if the database contains many small tables.

In an SMS table space, space for tables is allocated on demand. The amount of space that is allocated is dependent on the setting of the *multipage_alloc* database configuration parameter. If this configuration parameter is set to YES, then a full extent (typically made up of two or more pages) will be allocated when space is required. Otherwise, space will be allocated one page at a time.

Multipage file allocation is enabled by default. The value of the multipage_alloc database configuration parameter will indicate if multipage file allocation is enabled.

Note: Multipage file allocation is not applicable to temporary table spaces.

Multi-page file allocation only affects the data and index portions of a table. This means that the .LF, .LB, and .LBA files are not extended one extent at a time.

When all space in a single container in an SMS table space is allocated to tables, the table space is considered full, even if space remains in other containers. You can add containers to an SMS table space only on a database partition that does not yet have any containers.

Note: SMS table spaces can take advantage of file-system prefetching and caching.

SMS table spaces are defined using the MANAGED BY SYSTEM option on the CREATE DATABASE command, or on the CREATE TABLESPACE statement. You must consider two key factors when you design your SMS table spaces:

- Containers for the table space.

  You must specify the number of containers that you want to use for your table space. It is very important to identify all the containers you want to use, because you cannot add or delete containers after an SMS table space is created. In a partitioned database environment, when a new database partition is added to the database partition group for an SMS table space, the ALTER TABLESPACE statement can be used to add containers to the new database partition.

  Each container used for an SMS table space identifies an absolute or relative directory name. Each of these directories can be located on a different file system (or physical disk). The maximum size of the table space can be estimated by:

  ```
  number of containers * (maximum file system size
     supported by the operating system)
  ```

  This formula assumes that there is a distinct file system mapped to each container, and that each file system has the maximum amount of space available. In practice, this may not be the case, and the maximum table space size may be much smaller. There are also SQL limits on the size of database objects, which may affect the maximum size of a table space.

  Note: Care must be taken when defining the containers. If there are existing files or directories on the containers, an error (SQL0298N) is returned.

- Extent size for the table space.

The extent size can be specified only when the table space is created. Because it cannot be changed later, it is important to select an appropriate value for the extent size.

If you do not specify the extent size when creating a table space, the database manager will create the table space using the default extent size, defined by the *dft_extent_sz* database configuration parameter. This configuration parameter is initially set based on information provided when the database is created. If the *dft_extent_sz* parameter is not specified on the CREATE DATABASE command, the default extent size will be set to 32.

To choose appropriate values for the number of containers and the extent size for the table space, you must understand:

- The limitation that your operating system imposes on the size of a logical file system.

  For example, some operating systems have a 2 GB limit. Therefore, if you want a 64 GB table object, you will need at least 32 containers on this type of system.

  When you create the table space, you can specify containers that reside on different file systems and, as a result, increase the amount of data that can be stored in the database.
- How the database manager manages the data files and containers associated with a table space.

  The first table data file (`SQL00001.DAT`) is created in the first container specified for the table space, and this file is allowed to grow to the extent size. After it reaches this size, the database manager writes data to `SQL00001.DAT` in the next container. This process continues until all of the containers contain `SQL00001.DAT` files, at which time the database manager returns to the first container. This process (known as *striping*) continues through the container directories until a container becomes full (SQL0289N), or no more space can be allocated from the operating system (disk full error). Striping is also used for index (`SQL`*nnnnn*`.INX`), long field (`SQL`*nnnnn*`.LF`), LOB (`SQL`*nnnnn*`.LB` and `SQL`*nnnnn*`.LBA`) and XML (`SQL`*nnnnn*`.XDA`) files.

  Note: The SMS table space is full as soon as any one of its containers is full. Thus, it is important to have the same amount of space available to each container.

  To help distribute data across the containers more evenly, the database manager determines which container to use first by taking the table identifier (`SQL00001.DAT` in the above example) and factoring into account the number of containers. Containers are numbered sequentially, starting at 0.

Related concepts:

- "Comparison of SMS and DMS table spaces" on page 267
- "Database managed space" on page 265
- "Table space design" on page 257

Related reference:

- "db2empfa - Enable multipage file allocation command" in *Command Reference*
- "multipage_alloc - Multipage file allocation enabled configuration parameter" in *Performance Guide*

Database managed space

In a DMS (Database Managed Space) table space, the database manager controls the storage space. The storage model consists of a limited number of devices or files whose space is managed by DB2 Database for Linux, UNIX, and Windows. The database administrator decides which devices and files to use, and DB2 manages the space on those devices and files. The table space is essentially an implementation of a special purpose file system designed to best meet the needs of the database manager.

DMS table spaces are different from SMS table spaces in that space for DMS table spaces is allocated when the table space is created. For SMS table spaces, space is allocated as needed. A DMS table space containing user defined tables and data can be defined as a *regular* or *large* table space that stores any table data or index data.

When designing your DMS table spaces and containers, you should consider the following:

- The database manager uses striping to ensure an even distribution of data across all containers.
- The maximum size of a regular table space is 512 GB for 32 KB pages. The maximum size of a large table space is 16TB. See SQL and XQuery limits for the maximum size of regular table spaces for other page sizes.
- Unlike SMS table spaces, the containers that make up a DMS table space do not need to be the same size; however, this is not normally recommended, because it results in uneven striping across the containers, and sub-optimal performance. If any container is full, DMS table spaces use available free space from other containers.
- Because space is pre-allocated, it must be available before the table space can be created. When using device containers, the device must also exist with enough space for the definition of the container. Each device can have only one container defined on it. To avoid wasted space, the size of the device and the size of the container should be equivalent. If, for example, the device is allocated with 5 000 pages, and the device container is defined to allocate 3 000 pages, 2 000 pages on the device will not be usable.
- By default, one extent in every container is reserved for overhead. Only full extents are used, so for optimal space management, you can use the following formula to determine an appropriate size to use when allocating a container:

  ```
  extent_size * (n + 1)
  ```

  where *extent_size* is the size of each extent in the table space, and *n* is the number of extents that you want to store in the container.
- The minimum size of a DMS table space is five extents. Attempting to create a table space smaller than five extents will result in an error (SQL1422N).
  - Three extents in the table space are reserved for overhead.
  - At least two extents are required to store any user table data. (These extents are required for the regular data for one table, and not for any index, long field or large object data, which require their own extents.)
- Device containers must use logical volumes with a “character special interface,” not physical volumes.

- You can use files instead of devices with DMS table spaces. No operational difference exists between a file and a device; however, a file can be less efficient because of the run-time overhead associated with the file system. Files are useful when:
  - Devices are not directly supported
  - A device is not available
  - Maximum performance is not required
  - You do not want to set up devices.
- If your workload involves LOBs or LONG VARCHAR data, you may derive performance benefits from file system caching.

  Note: LOBs and LONG VARCHARs are not buffered by the database manager's buffer pool.
- Some operating systems allow you to have physical devices greater than 2 GB in size. You should consider dividing the physical device into multiple logical devices, so that no container is larger than the size allowed by the operating system.

Note: Like SMS table spaces, DMS file containers can take advantage of file system prefetching and caching. However, DMS table spaces that use raw device containers cannot.

There is one exception to this general statement regarding contiguous placement of pages in storage. There are two container options when working with DMS table spaces: raw devices and files. When working with file containers, the database manager allocates the entire container at table space creation time. A result of this initial allocation of the entire table space is that the physical allocation is typically, but not guaranteed to be, contiguous even though the file system is doing the allocation. When working with raw device containers, the database manager takes control of the entire device and always ensures the pages in an extent are contiguous.

When working with DMS table spaces, you should consider associating each container with a different disk. This allows for a larger table space capacity and the ability to take advantage of parallel I/O operations.

The CREATE TABLESPACE statement creates a new table space within a database, assigns containers to the table space, and records the table space definition and attributes in the catalog. When you create a table space, the extent size is defined as a number of contiguous pages. The extent is the unit of space allocation within a table space. Only one table or object, such as an index, can use the pages in any single extent. All objects created in the table space are allocated extents in a logical table space address map. Extent allocation is managed through Space Map Pages (SMP).

The first extent in the logical table space address map is a header for the table space containing internal control information. The second extent is the first extent of Space Map Pages (SMP) for the table space. SMP extents are spread at regular intervals throughout the table space. Each SMP extent is a bit map of the extents from the current SMP extent to the next SMP extent. The bit map is used to track which of the intermediate extents are in use.

The next extent following the SMP is the object table for the table space. The object table is an internal table that tracks which user objects exist in the table space and where their first Extent Map Page (EMP) extent is located. Each object has its own EMPs which provide a map to each page of the object that is stored in the logical table space address map. Figure 24 shows how extents are allocated in a logical table space address map.

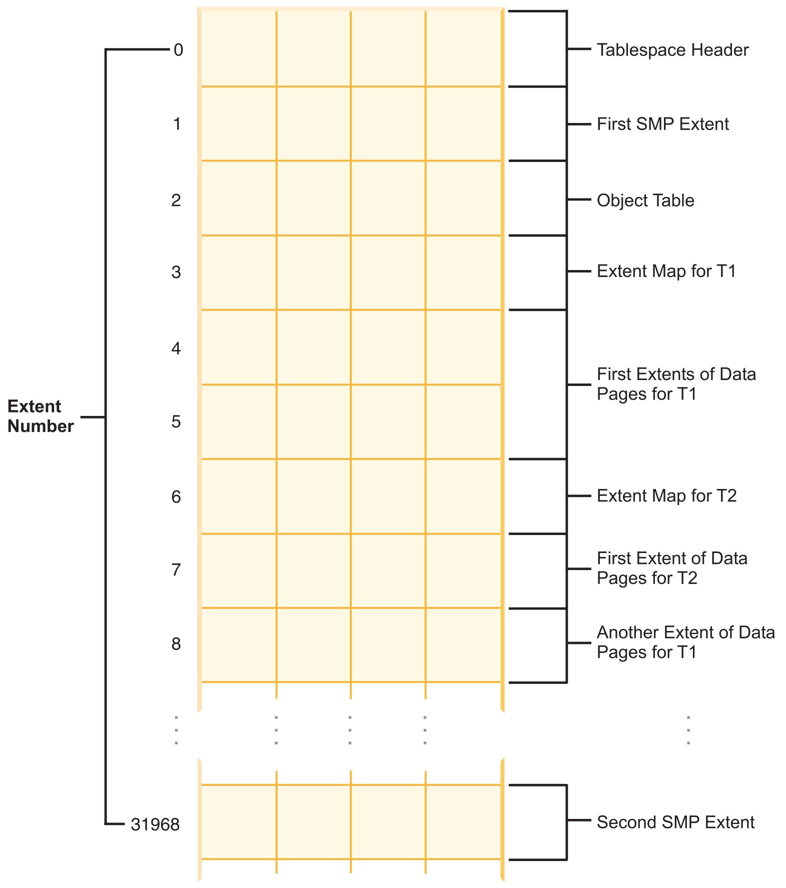


Figure 24. Logical table space address map

Related concepts:

- "Comparison of SMS and DMS table spaces" on page 267
- "How containers are added and extended in DMS table spaces" in *Administration Guide: Planning*
- "System managed space" on page 262
- "Table space design" on page 257
- "Table space maps" in *Administration Guide: Planning*

Comparison of SMS and DMS table spaces

There are a number of trade-offs to consider when determining which type of table space you should use to store your data.

Advantages of an SMS Table Space:

- Space is not allocated by the system until it is required.
- Creating a table space requires less initial work, because you do not have to predefine the containers.

- Indexes created on distributed data can be stored in a different table space than the table data.

Advantages of a DMS Table Space:

- The size of a table space can be increased by adding or extending containers, using the ALTER TABLESPACE statement. Existing data can be automatically rebalanced across the new set of containers to retain optimal I/O efficiency.
- A table can be split across multiple table spaces, based on the type of data being stored:
  - Long field and LOB data
  - Indexes
  - Regular table data

  You might want to separate your table data for performance reasons, or to increase the amount of data stored for a table. For example, you could have a table with 64 GB of regular table data, 64 GB of index data and 2 TB of long data. If you are using 8 KB pages, the table data and the index data can be as much as 128 GB. If you are using 16 KB pages, it can be as much as 256 GB. If you are using 32 KB pages, the table data and the index data can be as much as 512 GB.
- Indexes created on distributed data can be stored in a different table space than the table data.
- The location of the data on the disk can be controlled, if this is allowed by the operating system.
- If all table data is in a single table space, a table space can be dropped and redefined with less overhead than dropping and redefining a table.
- In general, a well-tuned set of DMS table spaces will outperform SMS table spaces.

Notes:

1. On the Solaris operating system, DMS table spaces with raw devices are strongly recommended for performance-critical workloads.
2. For performance-sensitive applications, particularly those involving a large number of insert operations, it is recommended that you use DMS table spaces.

Also, placement of data can differ on the two types of table spaces. For example, consider the need for efficient table scans: it is important that the pages in an extent are physically contiguous. With SMS, the file system of the operating system decides where each logical file page is physically placed. The pages might be allocated contiguously depending on the level of other activity on the file system and the algorithm used to determine placement. With DMS, however, the database manager can ensure the pages are physically contiguous because it interfaces with the disk directly.

In general, small personal databases are easiest to manage with SMS table spaces. On the other hand, for large, growing databases you will probably only want to use SMS table spaces for the temporary table spaces and catalog table space, and separate DMS table spaces, with multiple containers, for each table. In addition, you will probably want to store long field data and indexes on their own table spaces.

If you choose to use DMS table spaces with device containers, you must be willing to tune and administer your environment.

Related concepts:
- "Database managed space" on page 265
- "System managed space" on page 262
- "Table space design" on page 257

Relationship between table spaces and buffer pools

Each table space is associated with a specific buffer pool. The default buffer pool is IBMDEFAULTBP. If another buffer pool is to be associated with a table space, the buffer pool must exist (it is defined with the CREATE BUFFERPOOL statement), it must have the same page size, and the association is defined when the table space is created (using the CREATE TABLESPACE statement). The association between the table space and the buffer pool can be changed using the ALTER TABLESPACE statement.

Having more than one buffer pool allows you to configure the memory used by the database to improve overall performance. For example, if you have a table space with one or more large (larger than available memory) tables that are accessed randomly by users, the size of the buffer pool can be limited, because caching the data pages might not be beneficial. The table space for an online transaction application might be associated with a larger buffer pool, so that the data pages used by the application can be cached longer, resulting in faster response times. Care must be taken in configuring new buffer pools.

Note: If you have determined that a page size of 8 KB, 16 KB, or 32 KB is required by your database, each table space with one of these page sizes must be mapped to a buffer pool with the same page size.

The storage required for all the buffer pools must be available to the database manager when the database is started. If DB2 Database for Linux, UNIX, and Windows is unable to obtain the required storage, the database manager will start up with default buffer pools (one each of 4 KB, 8 KB, 16 KB, and 32 KB page sizes), and issue a warning.

In a partitioned database environment, you can create a buffer pool of the same size for all database partitions in the database. You can also create buffer pools of different sizes on different database partitions.

Related concepts:
- "Table spaces and other storage structures" on page 261

Related reference:
- "ALTER BUFFERPOOL statement" in *SQL Reference, Volume 2*
- "ALTER TABLESPACE " on page 898
- "CREATE BUFFERPOOL statement" in *SQL Reference, Volume 2*
- "CREATE TABLESPACE " on page 1021

Relationship between table spaces and database partition groups

In a partitioned database environment, each table space is associated with a specific database partition group. This allows the characteristics of the table space to be applied to each database partition in the database partition group. The database partition group must exist (it is defined with the CREATE DATABASE

PARTITION GROUP statement), and the association between the table space and the database partition group is defined when the table space is created using the CREATE TABLESPACE statement.

You cannot change the association between table space and database partition group using the ALTER TABLESPACE statement. You can only change the table space specification for individual database partitions within the database partition group. In a single-partition environment, each table space is associated with the default database partition group. The default database partition group, when defining a table space, is IBMDEFAULTGROUP, unless a system temporary table space is being defined; then IBMTEMPGROUP is used.

Related concepts:
- "Table spaces and other storage structures" on page 261
- "Database partition groups," on page 250
- "Table space design" on page 257

Related reference:
- "CREATE DATABASE PARTITION GROUP " on page 918
- "CREATE TABLESPACE " on page 1021

Temporary table space design

System temporary table spaces hold temporary data required by the database manager while performing operations such as sorts or joins. These types of operations require extra space to process the results set. A database must have at least one system temporary table space; by default, one system temporary table space called TEMPSPACE1 is created at database creation time. IBMTEMPGROUP is the default database partition group for this table space.

User temporary table spaces hold temporary data from tables created with a DECLARE GLOBAL TEMPORARY TABLE statement. To allow the definition of declared temporary tables, at least one user temporary table space should be created with the appropriate USE privileges. USE privileges are granted using the GRANT statement. A user temporary table spaces is *not* created by default at database creation time.

It is recommended that you define a single SMS temporary table space with a page size equal to the page size used in the majority of your regular table spaces. This should be suitable for typical environments and workloads. However, it can be advantageous to experiment with different temporary table space configurations and workloads. The following points should be considered:
- Temporary tables are in most cases accessed in batches and sequentially. That is, a batch of rows is inserted, or a batch of sequential rows is fetched. Therefore, a larger page size typically results in better performance, because fewer logical or physical page I/O requests are required to read a given amount of data. This is not always the case when the average temporary table row size is smaller than the page size divided by 255. A maximum of 255 rows can exist on any page, regardless of the page size. For example, a query that requires a temporary table with 15-byte rows would be better served by a 4 KB temporary table space page size, because 255 such rows can all be contained within a 4 KB page. An 8 KB (or larger) page size would result in at least 4 KB (or more) bytes of wasted space on each temporary table page, and would not reduce the number of required I/O requests.

- If more than fifty percent of the regular table spaces in your database use the same page size, it can be advantageous to define your temporary table spaces with the same page size. The reason for this is that this arrangement enables your temporary table space to share the same buffer pool space with most or all of your regular table spaces. This, in turn, simplifies buffer pool tuning.
- When reorganizing a table using a temporary table space, the page size of the temporary table space must match that of the table. For this reason, you should ensure that there are temporary table spaces defined for each different page size used by existing tables that you may reorganize using a temporary table space.

  You can also reorganize without a temporary table space by reorganizing the table directly in the target table space. Of course, this type of reorganization requires that there be extra space in the target table space for the reorganization process.
- If you are reliant on system temporary tables in SMS system temporary table spaces because of your work envionment, you may want to consider using the registry variable DB2_SMS_TRUNC_TMPTABLE_THRESH. In the past when system temporary tables were no longer needed, they were truncated to a file size of zero. The need for a new system temporary table would have a performance cost associated with it. Using this registry variable allows for leaving non-zero system temporary tables on the system to avoid the performance cost of repeated creations and truncations of system temporary tables.
- In general, when temporary table spaces of differing page sizes exist, the optimizer will choose the temporary table space whose buffer pool can hold the most number of rows (in most cases that means the largest buffer pool). In such cases, it is often wise to assign an ample buffer pool to one of the temporary table spaces, and leave any others with a smaller buffer pool. Such a buffer pool assignment will help ensure efficient utilization of main memory. For example, if your catalog table space uses 4 KB pages, and the remaining table spaces use 8 KB pages, the best temporary table space configuration may be a single 8 KB temporary table space with a large buffer pool, and a single 4 KB table space with a small buffer pool.
- There is generally no advantage to defining more than one temporary table space of any single page size.
- SMS is almost always a better choice than DMS for temporary table spaces because:
  - There is more overhead in the creation of a temporary table when using DMS versus SMS.
  - Disk space is allocated on demand in SMS, whereas it must be pre-allocated in DMS. Pre-allocation can be difficult: Temporary table spaces hold transient data that can have a very large peak storage requirement, and a much smaller average storage requirement. With DMS, the peak storage requirement must be pre-allocated, whereas with SMS, the extra disk space can be used for other purposes during off-peak hours.
  - The database manager attempts to keep temporary table pages in memory, rather than writing them out to disk. As a result, the performance advantages of DMS are less significant.

Related concepts:

- "System managed space" on page 262
- "Table space design" on page 257
- "Temporary tables in SMS table spaces" on page 272

Related reference:
- "REORG INDEXES/TABLE " on page 580

Temporary tables in SMS table spaces

Temporary tables in SMS table spaces are not deleted by default once they are no longer needed. Instead, files associated with temporary tables are truncated to a length of zero. In cases where temporary tables are used repeatedly, this avoids some of the performance cost of deleting and recreating temporary tables.

This reuse of temporary tables benefits users whose workload involves dealing with many small temporary tables on smaller systems such as Windows where the file system calls are relatively expensive; and users whose disk storage is distributed, requiring network messages to complete file system operations.

By default, files that hold temporary tables are truncated to a zero length or to the extent size specified in the DB2_SMS_TRUNC_TMPTABLE_THRESH registry variable once they are no longer needed. You can set the number of extents to be used by specifying a value for the DB2_SMS_TRUNC_TMPTABLE_THRESH registry variable. You should increase the value associated with this registry variable if your workload repeatedly uses large SMS temporary tables and you can afford to leave space allocated between uses.

You can turn off this feature by specifying a value of 0 for the DB2_SMS_TRUNC_TMPTABLE_THRESH registry variable. You might want to do this if your system has restrictive space limitations and you are experiencing repeated out of disk errors for SMS temporary table spaces.

The first connection to the database deletes any previously allocated files. If you want to clear out existing temporary tables, you should drop all database connections and reconnect, or deactivate the database and reactivate it. If you want to ensure that space for temporary tables stays allocated, use the ACTIVATE DATABASE command to start the database. This will avoid the repeated cost of startup on the first connect to the database.

Related concepts:
- "Temporary table space design" on page 270

Catalog table space design

An SMS table space is recommended for database catalogs, for the following reasons:
- The database catalog consists of many tables of varying sizes. When using a DMS table space, a minimum of two extents are allocated for each table object. Depending on the extent size chosen, a significant amount of allocated and unused space may result. When using a DMS table space, a small extent size (two to four pages) should be chosen; otherwise, an SMS table space should be used.
- There are large object (LOB) columns in the catalog tables. LOB data is not kept in the buffer pool with other data, but is read from disk each time it is needed. Reading LOBs from disk reduces performance. Since a file system usually has its own cache, using an SMS table space, or a DMS table space built on file containers, makes avoidance of I/O possible if the LOB has previously been referenced.

Given these considerations, an SMS table space is a somewhat better choice for the catalogs.

Another factor to consider is whether you will need to enlarge the catalog table space in the future. While some platforms have support for enlarging the underlying storage for SMS containers, and while you can use redirected restore to enlarge an SMS table space, the use of a DMS table space facilitates the addition of new containers.

Note: When creating a database, three table spaces are defined, including the SYSCATSPACE table space for the system catalog tables. The page size that becomes the default for all table spaces is set when the database is created. If a page size greater than 4096 (or 4 KB) is chosen, the page size for the catalog tables is restricted to a row size that it would have if the catalog table space had a page size of 4 KB. The default database page size is stored as an informational database configuration parameter called *pagesize*.

Related concepts:

- "Database managed space" on page 265
- "System managed space" on page 262
- "Table space design" on page 257
- "System catalog tables" on page 297

Chapter 23. Before Creating the Database

Starting a DB2 instance (Linux, UNIX)

You might need to start or stop the DB2 database during normal business operations; for example, you must start an instance before you can perform the following tasks:

- Connecting to a database on the instance
- Precompiling an application
- Binding a package to a database
- Accessing host databases.

Prerequisites:

Before you start a DB2 instance on your system:

1. Log in with a user ID or name that has SYSADM, SYSCTRL, or SYSMAINT authority on the instance; or log in as the instance owner.
2. Run the startup script as follows:
```
   . INSTHOME/sqllib/db2profile      (for Bourne or Korn shell)
   source INSTHOME/sqllib/db2cshrc   (for C shell)
```
   where INSTHOME is the home directory of the instance you want to use.

Procedure:

To start the instance using the Control Center:

1. Expand the object tree until you see the **Instances** folder.
2. Right-click the instance that you want to start, and select **start** from the pop-up menu.

To start the instance using the command line, enter:

```
db2start
```

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current instance. For more information, see Setting the current instance environment variables.

Related tasks:

- "Setting the current instance environment variables" in *Administration Guide: Implementation*

- "Starting a DB2 instance (Windows)" on page 276
- "Stopping an instance (Linux, UNIX)" on page 277

Starting a DB2 instance (Windows)

You might need to start or stop a DB2 instance during normal business operations; for example, you must start an instance before you can perform the following tasks:

- Connecting to a database on the instance
- Precompiling an application
- Binding a package to a database
- Accessing host databases.

Prerequisites:

In order to successfully launch the DB2 database instance as a service from `db2start`, the user account must have the correct privilege as defined by the Windows operating system to start a Windows service. The user account can be a member of the Administrators, Server Operators, or Power Users group.

Procedure:

To start the instance using the Control Center:

1. Expand the object tree until you see the **Instances** folder.
2. Right-click the instance that you want to start, and select **start** from the pop-up menu.

To start the instance using the command line, enter:

```
db2start
```

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current instance. For more information, see Setting the current instance environment variables.

The `db2start` command will launch the DB2 database instance as a Windows service. The DB2 database instance on Windows can still be run as a process by specifying the ″/D″ switch when invoking `db2start`. The DB2 database instance can also be started as a service using the Control Panel or ″NET START″ command.

When running in a partitioned database environment, each database partition server is started as a Windows service. You can not use the ″/D″ switch to start a DB2 instance as a process in a partitioned database environment.

Related tasks:

- "Setting the current instance environment variables" in *Administration Guide: Implementation*
- "Starting a DB2 instance (Linux, UNIX)" on page 275
- "Stopping an instance (Windows)" on page 279
- "Stopping an instance (Linux, UNIX)" on page 277

Grouping objects by schema

Database object names might be made up of a single identifier or they might be *schema-qualified objects* made up of two identifiers. The schema, or high-order part, of a schema-qualified object provides a means to classify or group objects in the database. When an object such as a table, view, alias, distinct type, function, index, package or trigger is created, it is assigned to a schema. This assignment is done either explicitly or implicitly.

Explicit use of the schema occurs when you use the high-order part of a two-part object name when referring to that object in a statement. For example, USER A issues a CREATE TABLE statement in schema C as follows:

```
CREATE TABLE C.X (COL1 INT)
```

Implicit use of the schema occurs when you do not use the high-order part of a two-part object name. When this happens, the CURRENT SCHEMA special register is used to identify the schema name used to complete the high-order part of the object name. The initial value of CURRENT SCHEMA is the authorization ID of the current session user. If you want to change this during the current session, you can use the SET SCHEMA statement to set the special register to another schema name.

Some objects are created within certain schemas and stored in the system catalog tables when the database is created.

In dynamic SQL and XQuery statements, a schema qualified object name implicitly uses the CURRENT SCHEMA special register value as the qualifier for unqualified object name references. In static SQL and XQuery statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified database object names.

Before creating your own objects, you need to consider whether you want to create them in your own schema or by using a different schema that logically groups the objects. If you are creating objects that will be shared, using a different schema name can be very beneficial.

Related concepts:
- “System catalog tables” on page 297

Related tasks:
- “Creating a schema” on page 304

Related reference:
- “CURRENT SCHEMA ” on page 1528
- “SET SCHEMA ” on page 1209

Stopping an instance (Linux, UNIX)

You might need to stop the current instance of the database manager.

Prerequisites:

To stop an instance on your system, you must do the following:

1. Log in or attach to an instance with a user ID or name that has SYSADM, SYSCTRL, or SYSMAINT authority on the instance; or, log in as the instance owner.
2. Display all applications and users that are connected to the specific database that you want to stop. To ensure that no vital or critical applications are running, list applications. You need SYSADM, SYSCTRL, or SYSMAINT authority for this.
3. Force all applications and users off the database. You require SYSADM or SYSCTRL authority to force users.

Restrictions:

The `db2stop` command can only be run at the server. No database connections are allowed when running this command; however, if there are any instance attachments, they are forced off before the instance is stopped.

Note: If command line processor sessions are attached to an instance, you must run the `terminate` command to end each session before running the `db2stop` command. The `db2stop` command stops the instance defined by the DB2INSTANCE environment variable.

Procedure:

To stop the instance using the Control Center:

1. Expand the object tree until you find the **Instances** folder.
2. Click each instance you want to stop.
3. Right-click any of the selected instances, and select **stop** from the pop-up menu.
4. On the Confirm stop window, click **OK**.

To stop the instance using the command line, enter:

```
db2stop
```

You can use the db2stop command to stop, or drop, individual database partitions within a partitioned database environment. When working in a partitioned database environment and you are attempting to drop a logical partition using

```
db2stop drop nodenum <0>
```

you must ensure that no users are attempting to access the database. If they are, you will receive an error message SQL6030N.

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current instance. For more information, see Setting the current instance environment variables.

Related tasks:

- "Setting the current instance environment variables" in *Administration Guide: Implementation*

Related reference:

- "db2stop - Stop DB2 command" in *Command Reference*
- "TERMINATE command" in *Command Reference*

Stopping an instance (Windows)

You might need to stop the current instance of the database manager.

Prerequisites:

To stop an instance on your system, you must do the following:

1. The user account stopping the DB2 database service must have the correct privilege as defined by the Windows operating system. The user account can be a member of the Administrators, Server Operators, or Power Users group.
2. Display all applications and users that are connected to the specific database that you want to stop. To ensure that no vital or critical applications are running, list applications. You need SYSADM, SYSCTRL, or SYSMAINT authority for this.
3. Force all applications and users off the database. You require SYSADM or SYSCTRL authority to force users.

Restrictions:

The `db2stop` command can only be run at the server. No database connections are allowed when running this command; however, if there are any instance attachments, they are forced off before the DB2 database service is stopped.

Note: If command line processor sessions are attached to an instance, you must run the `terminate` command to end each session before running the `db2stop` command. The `db2stop` command stops the instance defined by the DB2INSTANCE environment variable.

Procedure:

To stop an instance on your system, use one of the following methods:

- `db2stop`
- Stop the service using the Control Center

1. Expand the object tree until you find the **Instances** folder.
2. Click each instance you want to stop.
3. Right-click any of the selected instances, and select **Stop** from the pop-up menu.
4. On the Confirm Stop window, click **OK**.

- Stop using the "NET STOP" command.
- Stop the instance from within an application.

Recall that when you are using the DB2 database manager in a partitioned database environment, each database partition server is started as a service. Each service must be stopped.

Note: When you run commands to start or stop an instance's database manager, the DB2 database manager applies the command to the current instance. For more information, see Setting the current instance environment variables.

Related tasks:

- "Setting the current instance environment variables" in *Administration Guide: Implementation*

Related reference:

- "db2stop - Stop DB2 command" in *Command Reference*

Instance creation

An instance is a logical database manager environment where you catalog databases and set configuration parameters. Depending on your needs, you can create more than one instance on the same physical server providing a unique database server environment for each instance. You can use multiple instances to do the following:

- Use one instance for a development environment and another instance for a production environment.
- Tune an instance for a particular environment.
- Restrict access to sensitive information.
- Control the assignment of SYSADM, SYSCTRL, and SYSMAINT authority for each instance.
- Optimize the database manager configuration for each instance.
- Limit the impact of an instance failure. In the event of an instance failure, only one instance is affected. Other instances can continue to function normally.

Multiple instances will require:

- Additional system resources (virtual memory and disk space) for each instance.
- More administration because of the additional instances to manage.

The instance directory stores all information that pertains to a database instance. You cannot change the location of the instance directory once it is created. The directory contains:

- The database manager configuration file
- The system database directory
- The node directory
- The node configuration file (db2nodes.cfg)
- Any other files that contain debugging information, such as the exception or register dump or the call stack for the DB2 database processes.

Terminology:

Bit-width
: The number of bits used to address virtual memory: 32-bit and 64-bit are the most common. This term might be used to refer to the bit-width of an instance, application code, external routine code. 32-bit application means the same things as 32-bit width application.

32-bit DB2 instance
: A DB2 instance that contains all 32-bit binaries including 32-bit shared libraries and executables.

64-bit DB2 instance
: A DB2 instance that contains 64-bit shared libraries and executables, and also all 32-bit client application libraries (included for both client and server), and 32-bit external routine support (included only on a server instance).

You can:

- Create an instance.
- Drop an instance.
- Start an instance.
- Stop an instance.
- Attach to an instance.

On UNIX operating systems, the instance directory is located in the INSTHOME/sqllib directory, where INSTHOME is the home directory of the instance owner.

On Windows operating systems, the instance directory is located in the /sqllib sub-directory, in the directory where the DB2 database product was installed.

In a partitioned database environment, the instance directory is shared between all database partition servers belonging to the instance. Therefore, the instance directory must be created on a network share drive that all computers in the instance can access.

As part of your installation procedure, you create an initial instance of DB2 called “DB2”. On UNIX, the initial instance can be called anything you want within the naming rules guidelines. The instance name is used to set up the directory structure.

To support the immediate use of this instance, the following are set during installation:
- The environment variable DB2INSTANCE is set to “DB2”.
- The DB2 registry variable DB2INSTDEF is set to “DB2”.

On UNIX, the default can be called anything you want within the naming rules guidelines.

On Windows, the instance name is the same as the name of the service, so it should not conflict. No instance name should be the same as another servicename. You must have the correct authorization to create a service.

These settings establish “DB2” as the default instance. You can change the instance that is used by default, but first you have to create an additional instance.

Before using DB2, the database environment for each user must be updated so that it can access an instance and run the DB2 database programs. This applies to all users (including administrative users).

On UNIX operating systems, sample script files are provided to help you set the database environment. The files are: db2profile for Bourne or Korn shell, and db2cshrc for C shell. These scripts are located in the sqllib subdirectory under the home directory of the instance owner. The instance owner or any user belonging to the instance’s SYSADM group can customize the script for all users of an instance. Use sqllib/userprofile and sqllib/usercshrc to customize a script for each user.

The blank files sqllib/userprofile and sqllib/usercshrc are created during instance creation to allow you to add your own instance environment settings. The db2profile and db2cshrc files are overwritten during an instance update in a DB2 FixPak installation. If you do not want the new environment settings in the db2profile or db2cshrc scripts, you can override them using the corresponding *user* script, which is called at the end of the db2profile or db2cshrc script. During an instance migration (using the `db2imigr` command), the *user* scripts are copied over so that your environment modifications will still be in use.

The sample script contains statements to:
- Update a user's PATH by adding the following directories to the existing search path: the bin, adm, and misc subdirectories under the sqllib subdirectory of the instance owner's home directory.
- Set the DB2INSTANCE environment variable to the instance name.

Related concepts:
- "Multiple instances on a Linux or UNIX operating system" in *Administration Guide: Implementation*
- "Multiple instances on a Windows operating system" in *Administration Guide: Implementation*
- "" in *Administration Guide: Planning*
- "Table spaces (Visual Explain)" in *Administration Guide: Planning*
- "About databases" in *Administration Guide: Planning*
- "Visual Explain tool" in *Administration Guide: Planning*

Related tasks:
- "Adding instances" in *Administration Guide: Implementation*
- "Auto-starting instances" in *Administration Guide: Implementation*
- "Creating additional instances" in *Administration Guide: Implementation*
- "Listing instances" in *Administration Guide: Implementation*
- "Running multiple instances concurrently (Windows)" in *Administration Guide: Implementation*
- "Setting the current instance environment variables" in *Administration Guide: Implementation*
- "UNIX details when creating instances" on page 284
- "Windows details when creating instances" on page 285

Setting the DB2 environment automatically on UNIX

By default, the scripts that set up the database environment when you create an instance affect the user environment for the duration of the current session only. You can change the `.profile` file to enable it to run the `db2profile` script automatically when the user logs on using the Bourne or Korn shell. For users of the C shell, you can change the `.login` file to enable it to run the `db2shrc` script file.

Procedure:

Add one of the following statements to the `.profile` or `.login` script files:
- For users who share one version of the script, add:
```
. INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
source INSTHOME/sqllib/db2cshrc (for C shell)
```
  where `INSTHOME` is the home directory of the instance that you want to use.
- For users who have a customized version of the script in their home directory, add:

```
. USERHOME/db2profile         (for Bourne or Korn shell)
source USERHOME/db2cshrc      (in C shell)
```

where USERHOME is the home directory of the user.

Related tasks:
- "Setting the DB2 environment manually on UNIX" on page 283

Setting the DB2 environment manually on UNIX

Procedure:

To choose which instance you want to use, enter one of the following statements at a command prompt. The period (.) and the space are required.

- For users who share one version of the script, add:

  ```
  . INSTHOME/sqllib/db2profile    (for Bourne or Korn shell)
  source INSTHOME/sqllib/db2cshrc (for C shell)
  ```

  where INSTHOME is the home directory of the instance that you want to use.
- For users who have a customized version of the script in their home directory, add:

  ```
  . USERHOME/db2profile         (for Bourne or Korn shell)
  source USERHOME/db2cshrc      (in C shell)
  ```

  where USERHOME is the home directory of the user.

If you want to work with more than one instance at the same time, run the script for each instance that you want to use in separate windows. For example, assume that you have two instances called `test` and `prod`, and their home directories are `/u/test` and `/u/prod`.

In window 1:
- In Bourne or Korn shell, enter:

  ```
  . /u/test/sqllib/db2profile
  ```
- In C shell, enter:

  ```
  source /u/test/sqllib/db2cshrc
  ```

In window 2:
- In Bourne or Korn shell, enter:

  ```
  . /u/prod/sqllib/db2profile
  ```
- In C shell, enter:

  ```
  source /u/prod/sqllib/db2cshrc
  ```

Use window 1 to work with the `test` instance and window 2 to work with the `prod` instance.

Note: Enter the `which db2` command to ensure that your search path has been set up correctly. This command returns the absolute path of the CLP executable. Verify that it is located under the instance's `sqllib` directory.

Related tasks:
- "Setting the DB2 environment automatically on UNIX" on page 282

UNIX details when creating instances

When working with UNIX operating systems, the `db2icrt` command has the following optional parameters:

- –h or –?

  This parameter is used to display a help menu for the command.
- –d

  This parameter sets the debug mode for use during problem determination.
- –a AuthType

  This parameter specifies the authentication type for the instance. Valid authentication types are SERVER, SERVER_ENCRYPT, or CLIENT. If not specified, the default is SERVER, if a DB2 server is installed. Otherwise, it is set to CLIENT.

  Notes:

  1. The authentication type of the instance applies to all databases owned by the instance.
  2. On UNIX operating systems, the authentication type DCE is not a valid choice.
- –u FencedID

  This parameter is the user under which the fenced user-defined functions (UDFs) and stored procedures will execute. This is not required if you install a DB2 client. For other DB2 products, this is a required parameter.

  Note: FencedID might not be "root" or "bin".
- –p PortName

  This parameter specifies the TCP/IP service name or port number to be used. This value will then be set in the instance's database configuration file for every database in the instance.
- –s InstType

  Allows different types of instances to be created. Valid instance types are: `ese`, `wse`, `client`, and `standalone`.

Examples:

- To add an instance for a DB2 server, you can use the following command:
  ```
  db2icrt -u db2fenc1 db2inst1
  ```
- If you installed the DB2 Connect Enterprise Server Edition only, you can use the instance name as the Fenced ID also:
  ```
  db2icrt -u db2inst1 db2inst1
  ```
- To add an instance for a DB2 client, you can use the following command:
  ```
  db2icrt db2inst1 –s client –u fencedID
  ```

DB2 client instances are created when you want a workstation to connect to other database servers and you have no need for a local database on that workstation.

Related reference:

- "db2icrt - Create instance " on page 479

Windows details when creating instances

When working with the Windows operating systems, the `db2icrt` command has the following optional parameters:

- –s InstType

  Allows different types of instances to be created. Valid instance types are: `ese`, `wse`, `client`, and `standalone`.
- –p:InstProf_Path

  This is an optional parameter to specify a different instance profile path. If you do not specify the path, the instance directory is created under the `SQLLIB` directory, and given the shared name `DB2` concatenated to the instance name. Read and write permissions are automatically granted to everyone in the domain. Permissions can be changed to restrict access to the directory.

  If you do specify a different instance profile path, you must create a shared drive or directory. This will allow the opportunity for everyone in the domain to access the instance directory unless permissions have been changed.
- –u:username,password

  When creating a partitioned database environment, you must declare the domain/user account name and password of the DB2 service.
- –r:base_port,end_port

  This is an optional parameter to specify the TCP/IP port range for the fast communications manager (FCM). If you specify the TCP/IP port range, you must ensure that the port range is available on all computers in the partition database system.

The following example could be used, on DB2 Enterprise Server Edition for Windows:

```
db2icrt inst1 –s ese
   –p:\\computerA\db2mpp
   –u:<user account name>,<password> –r:9010,9015
```

Note: If you change the service account; that is, if you no longer use the default service created when the first instance was created during product installation, then you must grant the domain/user account name used to create the instance the following advanced rights:

- Act as a part of the operating system
- Create a token object
- Increase quota
- Log on as a service
- Replace a process level token
- Lock page in memory

The instance requires these user rights to access the shared drive, authenticate the user account, and run DB2 as a Windows service. The “Lock page in memory” right is needed for Address Windowing Extensions (AWE) support.

Related reference:

- “db2icrt - Create instance ” on page 479

License management

The management of licenses for your DB2 products is done primarily through the License Center within the Control Center of the online interface to the product. From the License Center you can check the license information, statistics, authorized users, and current users for each of the installed products.

When the Control Center cannot be used, the `db2licm` Licensed Management Tool command performs basic license functions. With this command, you are able to add, remove, list, and modify licenses and policies installed on your local system.

Related concepts:

- "Control Center overview" in *Administration Guide: Implementation*
- "License Center overview" in *Administration Guide: Implementation*

Related reference:

- "db2licm - License management tool command" in *Command Reference*

Additional considerations for partitioned database environments

Enabling database partitioning in a database

The decision to create a multi-partition database must be made before you create your database. As part of the database design decisions you make, you will have to determine if you should take advantage of the performance improvements database partitioning can offer.

Some of the considerations surrounding your decision to create a database in a partitioned database environment are made here.

When running in a partitioned database environment, you can create a database from any database partition that exists in the `db2nodes.cfg` file using the `CREATE DATABASE` command or the sqlecrea() application programming interface (API).

Before creating a multi-partition database, you must select which database partition will be the catalog partition for the database. You can then create the database directly from that database partition, or from a remote client that is attached to that database partition. The database partition to which you attach and execute the CREATE DATABASE command becomes the *catalog partition* for that particular database.

The catalog partition is the database partition on which all system catalog tables are stored. All access to system tables must go through this database partition. All federated database objects (for example, wrappers, servers, and nicknames) are stored in the system catalog tables at this database partition.

If possible, you should create each database in a separate instance. If this is not possible (that is, you must create more than one database per instance), you should spread the catalog partitions among the available database partitions. Doing this reduces contention for catalog information at a single database partition.

Note: You should regularly do a backup of the catalog partition and avoid putting user data on it (whenever possible), because other data increases the time required for the backup.

When you create a database, it is automatically created across all the database partitions defined in the db2nodes.cfg file.

When the first database in the system is created, a system database directory is formed. It is appended with information about any other databases that you create. When working on UNIX, the system database directory is sqldbdir and is located in the sqllib directory under your home directory, or under the directory where DB2 database was installed. When working on UNIX, this directory must reside on a shared file system, (for example, NFS on UNIX platforms) because there is only one system database directory for all the database partitions that make up the partitioned database environment. When working on Windows, the system database directory is located in the instance directory.

Also resident in the sqldbdir directory is the system intention file. It is called sqldbins, and ensures that the database partitions remain synchronized. The file must also reside on a shared file system since there is only one directory across all database partitions. The file is shared by all the database partitions making up the database.

Configuration parameters have to be modified to take advantage of database partitioning. Use the `GET DATABASE CONFIGURATION` and the `GET DATABASE MANAGER CONFIGURATION` commands to find out the values of individual entries in a specific database, or in the database manager configuration file. To modify individual entries in a specific database, or in the database manager configuration file, use the `UPDATE DATABASE CONFIGURATION` and the `UPDATE DATABASE MANAGER CONFIGURATION` commands respectively.

The database manager configuration parameters affecting a partitioned database environment include *conn_elapse*, *fcm_num_buffers*, *fcm_num_channels*, *max_connretries*, *max_coordagents*, *max_time_diff*, *num_poolagents*, and *stop_start_time*.

Related tasks:
- "Configuring DB2 with configuration parameters" on page 1493

Related reference:
- "CREATE DATABASE " on page 461
- "sqlecrea - Create database" on page 800

Creating a node configuration file

Procedure:

If your database is to operate in a partitioned database environment, you must create a node configuration file called db2nodes.cfg. This file must be located in the sqllib subdirectory of the home directory for the instance before you can start the database manager with parallel capabilities across multiple database partitions. The file contains configuration information for all database partitions in an instance, and is shared by all database partitions for that instance.

Windows Considerations

If you are using DB2 Enterprise Server Edition on Windows, the node configuration file is created for you when you create the instance. You should not attempt to create or modify the node configuration file manually. You can use the `db2ncrt` command to add a database partition server to an instance. You can use the `db2ndrop` command to drop a database partition server from an instance. You can use the `db2nchg` command to modify a database partition server configuration including moving the database partition server from one computer to another; changing the TCP/IP host name; or, selecting a different logical port or network name.

Note: You should not create files or directories under the sqllib subdirectory other than those created by the DB2 database manager to prevent the loss of data if an instance is deleted. There are two exceptions. If your system supports stored procedures, put the stored procedure applications in the function subdirectory under the sqllib subdirectory. The other exception is when user-defined functions (UDFs) have been created. UDF executables are allowed in the same directory.

The file contains one line for each database partition that belongs to an instance. Each line has the following format:

```
dbpartitionnum hostname [logical-port [netname]]
```

Tokens are delimited by blanks. The variables are:

dbpartitionnum

The database partition number, which can be from 0 to 999, uniquely defines a database partition. Database partition numbers must be in ascending sequence. You can have gaps in the sequence.

Once a database partition number is assigned, it cannot be changed. (Otherwise the information in the distribution map, which specifies how data is distributed, would be compromised.)

If you drop a database partition, its database partition number can be used again for any new database partition that you add.

The database partition number is used to generate a database partition name in the database directory. It has the format:

```
NODEnnnn
```

The *nnnn* is the database partition number, which is left-padded with zeros. This database partition number is also used by the `CREATE DATABASE` and `DROP DATABASE` commands.

hostname

The hostname of the IP address for inter-partition communications. Use the fully-qualified name for the hostname. The `/etc/hosts` file also should use the fully-qualified name. If the fully-qualified name is not used in the db2nodes.cfg file and in the `/etc/hosts` file, you might receive error message SQL30082N RC=3.

(There is an exception when `netname` is specified. In this situation, `netname` is used for most communications, with `hostname` only being used for `db2start`, `db2stop`, and `db2_all`.)

logical-port

This parameter is optional, and specifies the logical port number for the database partition. This number is used with the database manager instance name to identify a TCP/IP service name entry in the `etc/services` file.

The combination of the IP address and the logical port is used as a well-known address, and must be unique among all applications to support communications connections between database partitions.

For each `hostname`, one `logical-port` must be either 0 (zero) or blank (which defaults to 0). The database partition associated with this `logical-port` is the default node on the host to which clients connect. You can override this with the DB2NODE environment variable in `db2profile` script, or with the sqlesetc() API.

`netname`
This parameter is optional, and is used to support a host that has more than one active TCP/IP interface, each with its own hostname.

The following example shows a possible node configuration file for an RS/6000 SP system on which SP2EN1 has multiple TCP/IP interfaces, two logical partitions, and uses SP2SW1 as the DB2 database interface. It also shows the database partition numbers starting at 1 (rather than at 0), and a gap in the `dbpartitionnum` sequence:

Table 68. Database partition number example table.

dbpartitionnum	hostname	logical-port	netname
1	SP2EN1.mach1.xxx.com	0	SP2SW1
2	SP2EN1.mach1.xxx.com	1	SP2SW1
4	SP2EN2.mach1.xxx.com	0	
5	SP2EN3.mach1.xxx.com		

You can update the db2nodes.cfg file using an editor of your choice. (The exception is: an editor should not be used on Windows.) You must be careful, however, to protect the integrity of the information in the file, as database partitioning requires that the node configuration file is locked when you issue `db2start` and unlocked after `db2stop` ends the database manager. The `db2start` command can update the file, if necessary, when the file is locked. For example, you can issue `db2start` with the RESTART option or the ADDNODE option.

Note: If the `db2stop` command is not successful and does not unlock the node configuration file, issue `db2stop FORCE` to unlock it.

Related concepts:
- "Guidelines for stored procedures" on page 1295

Related reference:
- "CREATE DATABASE " on page 461
- "db2nchg - Change database partition server configuration " on page 485
- "db2ncrt - Add database partition server to an instance " on page 486
- "db2ndrop - Drop database partition server from an instance " on page 487
- "db2start - Start DB2 command" in *Command Reference*
- "db2stop - Stop DB2 command" in *Command Reference*
- "DROP DATABASE " on page 494

Fast communications manager (FCM) communications

In a partitioned database environment, most communication between database partitions is handled by the fast communications manager (FCM). To enable the FCM at a database partition and allow communication with other database partitions, you must create a service entry in the database partition's services file of the etc directory as shown below. The FCM uses the specified port to communicate. If you have defined multiple database partitions on the same host, you must define a range of ports as shown below.

Before attempting to manually configure memory for the fast communications manager (FCM), it is recommented that you start with the automatic setting, which is also the default setting, for the number of FCM Buffers (*fcm_num_buffers*) and for the number of FCM Channels (*fcm_num_channels*). Use the system monitor data for FCM activity to determine if this setting is appropriate.

Windows Considerations

If you are using DB2 Enterprise Server Edition in the Windows environment, the TCP/IP port range is automatically added to the services file by:

- The install program when it creates the instance or adds a new database partition
- The `db2icrt` utility when it creates a new instance
- The `db2ncrt` utility when it adds the first database partition on the computer

The syntax of a service entry is as follows:

```
DB2_instance port/tcp #comment
```

DB2_*instance*

The value for *instance* is the name of the database manager instance. All characters in the name must be lowercase. Assuming an instancev name of `db2puser`, you would specify `DB2_db2puser`

port/**tcp**

The TCP/IP port that you want to reserve for the database partition.

#*comment*

Any comment that you want to associate with the entry. The comment must be preceded by a pound sign (#).

If the `services` file of the `etc` directory is shared, you must ensure that the number of ports allocated in the file is either greater than or equal to the largest number of multiple database partitions in the instance. When allocating ports, also ensure that you account for any processor that can be used as a backup.

If the services file of the etc directory is not shared, the same considerations apply, with one additional consideration: you must ensure that the entries defined for the DB2 database instance are the same in all services files of the etc directory (though other entries that do not apply to your partitioned database environment do not have to be the same).

If you have multiple database partitions on the same host in an instance, you must define more than one port for the FCM to use. To do this, include two lines in the services file of the etc directory to indicate the range of ports you are allocating. The first line specifies the first port, while the second line indicates the end of the block of ports. In the following example, five ports are allocated for the instance `sales`. This means no processor in the instance has more than five database partitions. For example,

```
DB2_sales        9000/tcp
DB2_sales_END    9004/tcp
```

Note: You must specify `END` in uppercase only. Also you must ensure that you include both underscore (_) characters.

Due to the way the FCM infrastructure utilizes TCP sockets and directs network traffic, FCM users on AIX 5.x should set the kernel parameter ″tcp_nodelayack″ to 1.

Related concepts:

- “Database partition and processor environments” on page 42
- “Aggregate registry variables” in *Administration Guide: Implementation*
- “The FCM buffer pool and memory requirements” in *Performance Guide*

Related reference:

- “MPP configuration variables” in *Performance Guide*

FCM considerations for a Common Criteria compliant environment

When the Database Partitioning Feature (DPF) is used with multiple physical nodes (that is, two or more physically separate computers), communication between the physical nodes occurs over a network using TCP/IP. DB2 does not protect that communication (ie, it is not encrypted or otherwise secured). Therefore, DB2 should be installed and operated in an environment where such protections are provided by the hardware or operating system associated with the DB2 instance. For example, DPF-related network traffic could be isolated to a separate network (wholly contained in the same secure environment where the DB2 servers reside), or encrypted either via IPsec or through the use of a hardware based network encryption solution.

Chapter 24. Creating a Database and Database Objects

Creating a database

You can create a database using the `CREATE DATABASE` command.

When you create a database, each of the following tasks are done for you:
- Setting up of all the system catalog tables that are needed by the database
- Allocation of the database recovery log
- Creation of the database configuration file and the default values are set
- Binding of the database utilities to the database

The Configuration Advisor helps you to tune performance and to balance memory requirements for a single database per instance by suggesting which configuration parameters to modify and providing suggested values for them. In Version 9.1, the Configuration Advisor is automatically invoked when you create a database. To disable this feature, or to explicitly enable it, use the `db2set` command before creating the database. Examples:

```
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=NO
db2set DB2_ENABLE_AUTOCONFIG_DEFAULT=YES
```

See Automatic features enabled by default for other DB2 features that are enabled by default.

Prerequisites:

You should have spent sufficient time designing the contents, layout, potential growth, and use of your database before you create it.

The following database privileges are automatically granted to PUBLIC: CREATETAB, BINDADD, CONNECT, IMPLICIT_SCHEMA, and SELECT on the system catalog views.

Procedure:

To create a database using the Control Center:

1. Expand the object tree until you find the **Databases** folder.
2. Right-click the **Databases** folder, and select **Create —> Standard** or **Create —> With Automatic Maintenance** from the pop-up menu.
3. Follow the steps to complete this task.

To create a database from a client application, call the sqlecrea API.

To create a database using the command line processor, enter: `CREATE DATABASE` <database name>. For example, the following command line processor command creates a database called `personl`, in the default location, with the associated comment "Personnel DB for BSchiefer Co".

```
CREATE DATABASE personl
   WITH "Personnel DB for BSchiefer Co"
```

At the same time a database is created, a detailed deadlocks event monitor is also created. As with any monitor, there is some overhead associated with this event monitor. If you do not want the detailed deadlocks event monitor, then the event monitor can be dropped using the command:

```
DROP EVENT MONITOR db2detaildeadlock
```

To limit the amount of disk space that this event monitor consumes, the event monitor deactivates, and a message is written to the administration notification log, once it has reached its maximum number of output files. Removing output files that are no longer needed allows the event monitor to activate again on the next database activation.

You have the ability to create a database in a different, possibly remote, database manager instance. In this type of environment you have the ability to perform instance-level administration against an instance other than your default instance, including remote instances.

By default, databases are created in the code page of the application creating them. Therefore, if you create your database from a Unicode (UTF-8) client, your database will be created as a Unicode database. Similarly, if you create your database from an en_US (code page 819) client, your database will be created as a single byte US English database.

To override the default code page for the database, it is necessary to specify the desired code set and territory when creating the database. See the `CREATE DATABASE` CLP command or the `sqlecrea` API for information on setting the code set and territory.

In a future release of the DB2 database manager, the default code set will be changed to UTF-8 when creating a database, regardless of the application code page. If a particular code set and territory is needed for a database, then the code set and territory should be specified when the database is created.

Related concepts:

- "Additional database design considerations" in *Administration Guide: Planning*
- "Applications connected to Unicode databases" in *Developing SQL and External Routines*
- "Automatic features enabled by default" in *Administration Guide: Planning*
- "What to record in a database" in *Administration Guide: Planning*
- "Database authorities" on page 74
- "Multiple instances of the database manager" in *Administration Guide: Implementation*

Related tasks:

- "Converting non-Unicode databases to Unicode" in *Administration Guide: Planning*
- "Creating a Unicode database" in *Administration Guide: Planning*
- "Changing node and database configuration files" in *Administration Guide: Implementation*
- "Generating recommendations for database configuration" in *Administration Guide: Implementation*

Related reference:

- "sqlecrea - Create database" on page 800
- "CREATE DATABASE " on page 461

Initial database partition groups

When a database is initially created, database partitions are created for all database partitions specified in the db2nodes.cfg file. Other database partitions can be added or removed with the `ADD DBPARTITIONNUM` and `DROP DBPARTITIONNUM VERIFY` commands.

Three database partition groups are defined:

- IBMCATGROUP for the SYSCATSPACE table space, holding system catalog tables
- IBMTEMPGROUP for the TEMPSPACE1 table space, holding temporary tables created during database processing
- IBMDEFAULTGROUP for the USERSPACE1 table space, by default holding user tables and indexes.

Related concepts:

- "Database partition groups," on page 250

Related reference:

- "ADD DBPARTITIONNUM " on page 435
- "DROP DBPARTITIONNUM VERIFY " on page 495

Defining initial table spaces

When a database is created, three table spaces are defined:

- SYSCATSPACE for the system catalog tables
- TEMPSPACE1 for system temporary tables created during database processing
- USERSPACE1 for user-defined tables and indexes

Note: When you first create a database no user temporary table space is created.

If you do not specify any table space parameters with the `CREATE DATABASE` command, the database manager creates these table spaces using system managed storage (SMS) directory containers. These directory containers are created in the subdirectory created for the database. The extent size for these table spaces is set to the default.

If you do use the CREATE DATABASE command, you can specify the page size for the default buffer pool and the intial table spaces. This default also represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. If you do not specify the page size when creating the database, the default page size is 4 KB.

Prerequisites:

The database must be created and you must have the authority to create table spaces.

Procedure:

To define initial table spaces using the Control Center:

1. Expand the object tree until you see the **Databases** folder.
2. Right-click the **Databases** folder, and select **Create —> Standard** or **Create —> With Automatic Maintenance** from the pop-up menu.
3. Follow the steps to complete this task.

To define initial table spaces using the command line, enter:

```
CREATE DATABASE <name>
  CATALOG TABLESPACE
    MANAGED BY SYSTEM USING ('<path>')
    EXTENTSIZE <value> PREFETCHSIZE <value>
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE'<path>' 5000,
                               FILE'<path>' 5000)
    EXTENTSIZE <value> PREFETCHSIZE <value>
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('<path>')
  WITH "<comment>"
```

If you do not want to use the default definition for these table spaces, you might specify their characteristics on the CREATE DATABASE command. For example, the following command could be used to create your database on Windows:

```
CREATE DATABASE PERSONL
  CATALOG TABLESPACE
    MANAGED BY SYSTEM USING ('d:\pcatalog','e:\pcatalog')
    EXTENTSIZE 16 PREFETCHSIZE 32
  USER TABLESPACE
    MANAGED BY DATABASE USING (FILE'd:\db2data\personl' 5000,
                               FILE'd:\db2data\personl' 5000)
    EXTENTSIZE 32 PREFETCHSIZE 64
  TEMPORARY TABLESPACE
    MANAGED BY SYSTEM USING ('f:\db2temp\personl')
  WITH "Personnel DB for BSchiefer Co"
```

In this example, the definition for each of the initial table spaces is explicitly provided. You only need to specify the table space definitions for those table spaces for which you do not want to use the default definition.

Note: When working in a partitioned database environment, you cannot create or assign containers to specific database partitions. First, you must create the database with default user and temporary table spaces. Then you should use the CREATE TABLESPACE statement to create the required table spaces. Finally, you can drop the default table spaces.

The coding of the MANAGED BY phrase on the `CREATE DATABASE` command follows the same format as the MANAGED BY phrase on the CREATE TABLESPACE statement.

Related concepts:

- "System catalog tables" on page 297
- "Table space design" on page 257

Related tasks:

- "Creating a table space" on page 299

Related reference:

- "CREATE DATABASE " on page 461

System catalog tables

A set of system catalog tables is created and maintained for each database. These tables contain information about the definitions of the database objects (for example, tables, views, indexes, and packages), and security information about the type of access that users have to these objects. These tables are stored in the SYSCATSPACE table space.

These tables are updated during the operation of a database; for example, when a table is created. You cannot explicitly create or drop these tables, but you can query and view their content. When the database is created, in addition to the system catalog table objects, the following database objects are defined in the system catalog:

- A set of routines (functions and procedures) in the schemas SYSIBM, SYSFUN, and SYSPROC.
- A set of read-only views for the system catalog tables is created in the SYSCAT schema.
- A set of updatable catalog views is created in the SYSSTAT schema. These updatable views allow you to update certain statistical information to investigate the performance of a hypothetical database, or to update statistics without using the `RUNSTATS` utility.

After your database has been created, you might want to limit the access to the system catalog views.

Related concepts:

- "Catalog views" on page 161
- "Functions overview" on page 1263
- "User-defined functions" in *SQL Reference, Volume 1*

Related tasks:

- "Securing the system catalog view" on page 166

Related reference:

- "Functions" in *SQL Reference, Volume 1*

Database recovery log

A *database recovery log* keeps a record of all changes made to a database, including the addition of new tables or updates to existing ones. This log is made up of a number of *log extents*, each contained in a separate file called a *log file*.

The database recovery log can be used to ensure that a failure (for example, a system power outage or application error) does not leave the database in an inconsistent state. In case of a failure, the changes already made but not committed are rolled back, and all committed transactions, which might not have been physically written to disk, are redone. These actions ensure the integrity of the database.

Related concepts:

- "Understanding recovery logs" on page 1534

Binding utilities to the database

When a database is created, the database manager attempts to bind the utilities in db2ubind.lst to the database. This file is stored in the bnd subdirectory of your sqllib directory.

Binding a utility creates a *package*, which is an object that includes all the information needed to process specific SQL and XQuery statements from a single source file.

Note: If you want to use these utilities from a client, you must bind them explicitly.

Procedure:

To bind or rebind the utilities to a database, issue the following commands using the command line processor:

```
connect to sample
bind @db2ubind.lst
```

Note: You must be in the directory where these files reside to create the packages in the `sample` database. The bind files are found in the bnd subdirectory of the sqllib directory. In this example, `sample` is the name of the database.

Related tasks:

- "Creating a database" on page 293

Related reference:

- "BIND" on page 441

Creating database partition groups

You create a database partition group with the CREATE DATABASE PARTITION GROUP statement. This statement specifies the set of database partitions on which the table space containers and table data are to reside. This statement also:

- Creates a distribution map for the database partition group.
- Generates a distribution map ID.

- Inserts records into the following catalog tables:
  - SYSCAT.DBPARTITIONGROUPS
  - SYSCAT.PARTITIONMAPS
  - SYSCAT.DBPARTITIONGROUPDEF

Prerequisites:

The computers and systems must be available and capable of handling a partitioned database environment. You have purchased and installed DB2 Enterprise Server Edition. The database must exist.

Procedure:

To create a database partition group using the Control Center:

1. Expand the object tree until you see the **Database partition groups** folder.
2. Right-click the **Database partition groups** folder, and select **Create** from the pop-up menu.
3. On the Create Database partition groups window, complete the information, use the arrows to move nodes from the **Available nodes** box to the **Selected database partitions** box, and click **OK**.

To create a database partition group using the command line, enter:

```
CREATE DATABASE PARTITION GROUP <name> ON PARTITIONS (<value>,<value>)
```

For example, assume that you want to load some tables on a subset of the database partitions in your database. You would use the following command to create a database partition group of two database partitions (1 and 2) in a database consisting of at least three (0 to 2) database partitions:

```
CREATE DATABASE PARTITION GROUP mixng12 ON PARTITIONS (1,2)
```

The `CREATE DATABASE` command or sqlecrea() API also create the default system database partition groups, IBMDEFAULTGROUP, IBMCATGROUP, and IBMTEMPGROUP.

Related concepts:
- "Database partition groups," on page 250
- "Distribution maps" on page 252

Related reference:
- "CREATE DATABASE " on page 461
- "CREATE DATABASE PARTITION GROUP " on page 918
- "sqlecrea - Create database" on page 800

Creating a table space

Table spaces establish the relationship between the physical storage devices used by your database system and the logical containers or tables used to store data.

Creating a table space within a database assigns containers to the table space and records its definitions and attributes in the database system catalog. You can then create tables within this table space.

When you create a database, three initial table spaces are created. The page size for the three initial table spaces is based on the default that is established or accepted when you use the CREATE DATABASE command. This default also represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. If you do not specify the page size when creating the database, the default page size is 4 KB. If you do not specify the page size when creating a table space, the default page size is the one set when you created the database.

Prerequisites:

You must know the device or file names of the containers that you will reference when creating your table spaces. In addition, you must know the space associated with each device or file name that you will allocate to the table space.

Procedure:

To create a table space using the Control Center:

1. Expand the object tree until you see the **Table spaces** folder.
2. Right-click the **Table spaces** folder, and select **Create —> Table Space Using Wizard** from the pop-up menu.
3. Follow the steps in the wizard to complete your task.

To create an SMS table space using the command line, enter:

```
CREATE TABLESPACE <NAME>
   MANAGED BY SYSTEM
   USING ('<path>')
```

To create a DMS table space using the command line, enter:

```
CREATE TABLESPACE <NAME>
   MANAGED BY DATABASE
   USING (FILE'<path>' <size>)
```

The following SQL statement creates an SMS table space on Windows using three directories on three separate drives:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY SYSTEM
   USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
```

The following SQL statement creates a DMS table space using two file containers, each with 5,000 pages:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY DATABASE
   USING (FILE'd:\db2data\acc_tbsp' 5000,
          FILE'e:\db2data\acc_tbsp' 5000)
```

In the previous two examples, explicit names are provided for the containers. However, if you specify relative container names, the container is created in the subdirectory created for the database.

When creating table space containers, the database manager creates any directory levels that do not exist. For example, if a container is specified as `/project/user_data/container1`, and the directory `/project` does not exist, then the database manager creates the directories `/project` and `/project/user_data`.

Starting with DB2 Universal Database Version 8.2, FixPak 4, any directories created by the database manager are created with PERMISSION 700. This means that only the owner has read, write, and execute access.

When creating multiple instances, note the following scenario:

1. Using the same directory structure as above, suppose that directory levels `/project/user_data` do not exist.
2. user1 creates an instance, named `user1` by default, then creates a database, and then creates a table space with `/project/user_data/container1` as one of its containers.
3. user2 creates an instance, named `user2` by default, then creates a database, and then attempts to create a table space with `/project/user_data/container2` as one of its containers.

Because the database manager created directory levels `/project/user_data` with PERMISSION 700 from the first request, user2 does not have access to these directory levels and cannot create `container2` in those directories. In this case, the CREATE TABLESPACE operation fails.

There are two methods to resolve this conflict:

1. Create the directory `/project/user_data` before creating the table spaces and set the permission to whatever access is needed for both user1 and user2 to create the table spaces. If all levels of table space directory exist, the database manager does not modify the access.
2. After user1 creates `/project/user_data/container1`, set the permission of `/project/user_data` to whatever access is needed for user2 to create the table space.

If a subdirectory is created by the database manager, it might also be deleted by the database manager when the table space is dropped.

The assumption in the previous examples is that the table spaces are not associated with a specific database partition group. The default database partition group IBMDEFAULTGROUP is used when the following parameter is not specified in the statement:

```
IN database_partition_group_name
```

The following SQL statement creates a DMS table space on a Linux and UNIX system using three logical volumes of 10 000 pages each, and specifies their I/O characteristics:

```
CREATE TABLESPACE RESOURCE
   MANAGED BY DATABASE
   USING (DEVICE '/dev/rdblv6' 10000,
          DEVICE '/dev/rdblv7' 10000,
          DEVICE '/dev/rdblv8' 10000)
   OVERHEAD 7.5
   TRANSFERRATE 0.06
```

The UNIX devices mentioned in this SQL statement must already exist, and the instance owner and the SYSADM group must be able to write to them.

The following example creates a DMS table space on a database partition group called ODDGROUP in a UNIX multi-partition database. ODDGROUP must be previously created with a CREATE DATABASE PARTITION GROUP statement. In this case, the ODDGROUP database partition group is assumed to be made up of database partitions numbered 1, 3, and 5. On all database partitions, use the device /dev/hdisk0 for 10 000 4 KB pages. In addition, declare a device for each database partition of 40 000 4 KB pages.

```
CREATE TABLESPACE PLANS IN ODDGROUP
   MANAGED BY DATABASE
   USING (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n1hd01' 40000)
          ON DBPARTITIONNUM 1
         (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n3hd03' 40000)
          ON DBPARTITIONNUM 3
         (DEVICE '/dev/HDISK0' 10000, DEVICE '/dev/n5hd05' 40000)
          ON DBPARTITIONNUM 5
```

UNIX devices are classified into two categories: character serial devices and block-structured devices. For all file-system devices, it is normal to have a corresponding character serial device (or *raw* device) for each block device (or *cooked* device). The block-structured devices are typically designated by names similar to “hd0” or “fd0”. The character serial devices are typically designated by names similar to “rhd0”, “rfd0”, or “rmt0”. These character serial devices have faster access than block devices. The character serial device names should be used on the CREATE TABLESPACE command and not block device names.

The overhead and transfer rate help to determine the best access path to use when the SQL statement is compiled. The current defaults for new table spaces in databases created in DB2 Version 9.1 or later are:
- OVERHEAD 7.5 ms
- TRANSFERRATE 0.06 ms

New table spaces in databases created in earlier versions of DB2 use the following defaults:
- OVERHEAD 12.67 ms
- TRANSFERRATE 0.18 ms

DB2 can greatly improve the performance of sequential I/O using the sequential prefetch facility, which uses parallel I/O.

You can also create a table space that uses a page size larger than the default 4 KB size. The following SQL statement creates an SMS table space on a Linux and UNIX system with an 8 KB page size.

```
CREATE TABLESPACE SMS8K
   PAGESIZE 8192
   MANAGED BY SYSTEM
   USING ('FSMS_8K_1')
   BUFFERPOOL BUFFPOOL8K
```

Notice that the associated buffer pool must also have the same 8 KB page size.

The created table space cannot be used until the buffer pool it references is activated.

You can use the ALTER TABLESPACE SQL statement to add, drop, or resize containers to a DMS table space and modify the PREFETCHSIZE, OVERHEAD, and TRANSFERRATE settings for a table space. You should commit the transaction issuing the table space statement as soon as possible following the ALTER TABLESPACE SQL statement to prevent system catalog contention.

Note: The PREFETCHSIZE value should be a multiple of the EXTENTSIZE value. For example if the EXTENTSIZE is 10, the PREFETCHSIZE should be 20 or 30. You should use the following equation to set your prefetch size manually when creating a table space:

```
prefetch size = (number of containers) X (number of physical spindles per
                         container) X extent size
```

You should also consider letting the DB2 database system automatically determine the prefetch size.

Direct I/O (DIO) improves memory performance because it bypasses caching at the file system level. This process reduces CPU overhead and makes more memory available to the database instance.

Concurrent I/O (CIO) includes the advantages of DIO and also relieves the serialization of write accesses.

DIO and CIO are supported on AIX; DIO is supported on HP-UX, Solaris Operating Environment, Linux, and Windows operating systems.

The keywords NO FILE SYSTEM CACHING and FILE SYSTEM CACHING are part of the CREATE and ALTER TABLESPACE SQL statements to allow you to specify whether DIO or CIO is to be used with each table space. When NO FILE SYSTEM CACHING is in effect, the database manager attempts to use Concurrent I/O (CIO) wherever possible. In cases where CIO is not supported (for example, if JFS is used), DIO is used instead.

When you issue the CREATE TABLESPACE statement, the dropped table recovery feature is turned on by default. This feature lets you recover dropped table data using table space-level restore and rollforward operations. This is useful because it is faster than database-level recovery, and your database can remain available to users.

However, the dropped table recovery feature can have some performance impact on forward recovery when there are many drop table operations to recover or when the history file is very large.You might want to disable this feature if you plan to run numerous drop table operations, and you either uses circular logging or you do not think you will want to recover any of the dropped tables. To disable this feature, you can explicitly set the DROPPED TABLE RECOVERY option to OFF when you issue the CREATE TABLESPACE statement. Alternatively, you can turn off the dropped table recovery feature for an existing table space using the ALTER TABLESPACE statement.

Related concepts:

- “Table space design” on page 257
- “Table spaces in database partition groups” on page 304
- “Database managed space” on page 265
- “System managed space” on page 262
- “Sequential prefetching” in *Performance Guide*

Related tasks:

- “Recovering a dropped table” in *Data Recovery and High Availability Guide and Reference*

- “Enabling large page support in a 64-bit environment (AIX)” in *Administration Guide: Planning*

Related reference:
- “ALTER TABLESPACE ” on page 898
- “CREATE TABLESPACE ” on page 1021

Table spaces in database partition groups

By placing a table space in a multiple-partition database partition group, all of the tables within the table space are divided or partitioned across each database partition in the database partition group. The table space is created into a database partition group. Once in a database partition group, the table space must remain there; it cannot be changed to another database partition group. The CREATE TABLESPACE statement is used to associate a table space with a database partition group.

Related reference:
- “CREATE TABLESPACE ” on page 1021

Creating a schema

While organizing your data into tables, it might also be beneficial to group tables and other related objects together. This is done by defining a schema through the use of the CREATE SCHEMA statement. Information about the schema is kept in the system catalog tables of the database to which you are connected. As other objects are created, they can be placed within this schema.

Schemas might also be implicitly created when a user has IMPLICIT_SCHEMA authority. With this authority, users implicitly create a schema whenever they create an object with a schema name that does not already exist.

Unqualified access to objects within a schema is not allowed since the schema is used to enforce uniqueness in the database. This becomes clear when considering the possibility that two users could create two tables (or other objects) with the same name. Without a schema to enforce uniqueness, ambiguity would exist if a third user attempted to query the table. It is not possible to determine which table to use without some further qualification.

The definer of any objects created as part of the CREATE SCHEMA statement is the schema owner. This owner can GRANT and REVOKE schema privileges to other users.

To allow another user to access a table without entering a schema name as part of the qualification on the table name requires that a view be established for that user. The definition of the view would define the fully-qualified table name including the user’s schema; the user would simply need to query using the view name. The view would be fully-qualified by the user’s schema as part of the view definition.

Prerequisites:

The database tables and other related objects that are to be grouped together must exist.

To issue the `CREATE SCHEMA` statement, you must have DBADM authority.

To create a schema with any valid name, you need SYSADM or DBADM authority.

Restrictions:

If users do not have IMPLICIT_SCHEMA or DBADM authority, the only schema they can create is one that has the same name as their own authorization ID.

The new schema name cannot already exist in the system catalogs and it cannot begin with ″SYS″.

Procedure:

If a user has SYSADM or DBADM authority, then the user can create a schema with any valid name. When a database is created, IMPLICIT_SCHEMA authority is granted to PUBLIC (that is, to all users).

To create a schema using the Control Center:

1. Expand the object tree until you see the **Schema** folder within a database.
2. Right-click the **Schema** folder, and click **Create**.
3. Complete the information for the new schema, and click **OK**.

To create a schema using the command line, enter:

```
CREATE SCHEMA <name> AUTHORIZATION <name>
```

The following is an example of a CREATE SCHEMA statement that creates a schema for an individual user with the authorization ID ″joe″:

```
CREATE SCHEMA joeschma AUTHORIZATION joe
```

Related concepts:
- “Implicit schema authority (IMPLICIT_SCHEMA) considerations” on page 77
- “Grouping objects by schema” on page 277
- “Schema privileges” on page 78

Related tasks:
- “Setting a schema” on page 305

Related reference:
- “CREATE SCHEMA ” on page 948

Setting a schema

Once you have several schemas in existence, you might want to designate one as the default schema for use by unqualified object references in dynamic SQL and XQuery statements issued from within a specific DB2 connection.

Procedure:

To establish a default schema: Set the special register CURRENT SCHEMA to the schema you want to use as the default. For example:

```
SET CURRENT SCHEMA = 'SCHEMA01'
```

This statement can be used from within an application program or issued interactively. Once set, the value of the CURRENT SCHEMA special register is used as the qualifier (schema) for unqualified object references in dynamic SQL and XQuery statements, with the exception of the CREATE SCHEMA statement where an unqualified reference to a database object exists.

The initial value of the CURRENT SCHEMA special register is equal to the authorization ID of the current session user.

Related concepts:
- "Schemas" on page 69

Related reference:
- "CURRENT SCHEMA " on page 1528
- "Reserved schema names and reserved words" on page 29
- "SET SCHEMA " on page 1209

Creating and populating a table

Tables are the main repository of data within databases. Creating the tables and entering data to fill the tables will occur when you are creating an new database.

Prerequisites:

You must take the time to design and organize the tables that will hold your data.

Procedure:

After you determine how to organize your data into tables, the next step is to create those tables, by using the CREATE TABLE statement. The table descriptions are stored in the system catalog of the database to which you are connected.

To create a table using the Control Center:

1. Expand the object tree until you see the **Tables** folder.
2. Right-click the **Tables** folder, and click **Create**.
3. Follow the steps in the wizard to complete your tasks.

To create a table using the command line, enter:

```
CREATE TABLE <NAME>
   (<column_name>  <data_type>  <null_attribute>)
   IN <TABLE_SPACE_NAME)
```

The following is an example of a CREATE TABLE statement that creates the EMPLOYEE table in the RESOURCE table space. This table is defined in the sample database:

```
CREATE TABLE EMPLOYEE
   (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
    FIRSTNME  VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3),
    PHONENO   CHAR(4),
    PHOTO     BLOB(10M)   NOT NULL)
IN RESOURCE
```

When creating a table, you can choose to have the columns of the table based on the attributes of a structured type. Such a table is called a "typed table".

A typed table can be defined to inherit some of its columns from another typed table. Such a table is called a "subtable", and the table from which it inherits is called its "supertable". The combination of a typed table and all its subtables is called a "table hierarchy". The topmost table in the table hierarchy (the one with no supertable) is called the "root table" of the hierarchy.

To declare a global temporary table, use the DECLARE GLOBAL TEMPORARY TABLE statement.

You can also create a table that is defined based on the result of a query. This type of table is called a *materialized query table*.

Refer to the topics in the related information sections for other options that you should consider when creating and populating a table.

Related concepts:
- "Import Overview" in *Data Movement Utilities Guide and Reference*
- "Load overview" in *Data Movement Utilities Guide and Reference*
- "Moving data across platforms - file format considerations" in *Data Movement Utilities Guide and Reference*
- "Comparing IDENTITY columns and sequences" in *Administration Guide: Implementation*
- "Large object (LOB) column considerations" on page 308
- "Table creation" in *Administration Guide: Implementation*
- "User-defined types (UDTs)" in *Administration Guide: Implementation*

Related tasks:
- "Creating a hierarchy table or a typed table" in *Administration Guide: Implementation*
- "Creating a materialized query table" in *Administration Guide: Implementation*
- "Creating a sequence" in *Administration Guide: Implementation*
- "Creating a table in a partitioned database environment" on page 309
- "Creating a table in multiple table spaces" in *Administration Guide: Implementation*
- "Creating a user-defined temporary table" in *Administration Guide: Implementation*
- "Defining a generated column on a new table" in *Administration Guide: Implementation*
- "Defining an identity column on a new table" in *Administration Guide: Implementation*
- "Defining dimensions on a table" in *Administration Guide: Implementation*
- "Defining a unique constraint on a table" in *Administration Guide: Implementation*

Related reference:
- "CREATE TABLE " on page 956

- “DECLARE GLOBAL TEMPORARY TABLE statement” in *SQL Reference, Volume 2*
- “INSERT ” on page 1109
- “IMPORT ” on page 512
- “LOAD ” on page 542

Large object (LOB) column considerations

Before creating a table that contains large object columns, you need to answer the following questions:

1. Do you want to log changes to LOB columns?

   If you do not want to log these changes, you must turn logging off by specifying the NOT LOGGED clause when you create the table. For example:

   ```
   CREATE TABLE EMPLOYEE
      (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
       FIRSTNME  VARCHAR(12) NOT NULL,
       MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
       LASTNAME  VARCHAR(15) NOT NULL,
       WORKDEPT  CHAR(3),
       PHONENO   CHAR(4),
       PHOTO     BLOB(10M)   NOT NULL  NOT LOGGED)
   IN RESOURCE
   ```

   If the LOB column is larger than 1 GB, logging must be turned off. (As a rule of thumb, you might not want to log LOB columns larger than 10 MB.) As with other options specified on a column definition, the only way to change the logging option is to re-create the table.

   Even if you choose not to log changes, LOB columns are *shadowed* to allow changes to be rolled back, whether the roll back is the result of a system generated error, or an application request. Shadowing is a recovery technique in which current storage page contents are never overwritten. That is, old, unmodified pages are kept as “shadow” copies. These copies are discarded when they are no longer needed to support a transaction rollback.

   Note: When recovering a database using the RESTORE and ROLLFORWARD commands, LOB data that was “NOT LOGGED”and was written since the last backup will be *replaced by binary zeros*.

2. Do you want to minimize the space required for the LOB column?

   You can make the LOB column as small as possible using the COMPACT clause on the CREATE TABLE statement. For example:

   ```
   CREATE TABLE EMPLOYEE
      (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
       FIRSTNME  VARCHAR(12) NOT NULL,
       MIDINIT   CHAR(1)     NOT NULL WITH DEFAULT,
       LASTNAME  VARCHAR(15) NOT NULL,
       WORKDEPT  CHAR(3),
       PHONENO   CHAR(4),
       PHOTO     BLOB(10M)   NOT NULL  NOT LOGGED  COMPACT)
   IN RESOURCE
   ```

   There is a *performance cost* when appending to a table with a compact LOB column, particularly if the size of LOB values are increased (because of storage adjustments that must be made).

   Note: When moving LOBs, small LOBs are stored in the applications heap and large LOBs are stored in temporary tables within a 4 KB page size temporary table space.

On platforms where sparse file allocation is not supported and where LOBs are placed in SMS table spaces, consider using the COMPACT clause. Sparse file allocation has to do with how physical disk space is used by an operating system. An operating system that supports sparse file allocation does not use as much physical disk space to store LOBs as compared to an operating system not supporting sparse file allocation. The COMPACT option allows for even greater physical disk space "savings" regardless of the support of sparse file allocation. Because you can get some physical disk space savings when using COMPACT, you should consider using COMPACT if your operating system does not support sparse file allocation.

Note: DB2 Version 8 and later: System catalogs that use LOB columns and might take up more space than in previous versions.

3. Do you want better performance for LOB columns, including those LOB columns in the system catalogs?

   There are large object (LOB) columns in the catalog tables. LOB data is not kept in the buffer pool with other data but is read from disk each time it is needed. Reading from disk slows down the performance of DB2 where the LOB columns of the catalogs are involved. Since a file system usually has its own place for storing (or caching) data, using a SMS table space, or a DMS table space built on file containers, make avoidance of I/O possible when the LOB has previously been referenced.

Related concepts:

- "Space requirements for large object data" on page 245

Related reference:

- "Large objects (LOBs)" in *SQL Reference, Volume 1*
- "CREATE TABLE " on page 956

Creating a table in a partitioned database environment

There are performance advantages to creating a table across several database partitions in a partitioned database environment. The work associated with the retrieval of data can be divided among the database partitions.

Creating a table that will be a part of several database partitions is specified when you are creating the table. There is an additional option when creating a table in a partitioned database environment: the *distribution key*. A distribution key is a key that is part of the definition of a table. It determines the database partition on which each row of data is stored.

If you do not specify the distribution key explicitly, the following defaults are used. *Ensure that the default distribution key is appropriate.*

- If a primary key is specified in the CREATE TABLE statement, the first column of the primary key is used as the distribution key.
- If there is no primary key, the first column that is not a long field is used.
- If no columns satisfy the requirements for a default distribution key, the table is created without one (this is allowed only in single-partition database partition groups).

Prerequisites:

Before creating a table that will be physically divided or distributed, you need to consider the following:

- Table spaces can span more than one database partition. The number of database partitions they span depends on the number of database partitions in a database partition group.
- Tables can be collocated by being placed in the same table space or by being placed in another table space that, together with the first table space, is associated with the same database partition group.

Restrictions:

You must be careful to select an appropriate distribution key because *it cannot be changed later*. Furthermore, any unique indexes (and therefore unique or primary keys) must be defined as a superset of the distribution key. That is, if a distribution key is defined, unique keys and primary keys must include all of the same columns as the distribution key (they might have more columns).

The size limit for one database partition of a table is 64 GB, or the available disk space, whichever is smaller. (This assumes a 4 KB page size for the table space.) The size of the table can be as large as 64 GB (or the available disk space) times the number of database partitions. If the page size for the table space is 8 KB, the size of the table can be as large as 128 GB (or the available disk space) times the number of database partitions. If the page size for the table space is 16 KB, the size of the table can be as large as 256 GB (or the available disk space) times the number of database partitions. If the page size for the table space is 32 KB, the size of the table can be as large as 512 GB (or the available disk space) times the number of database partitions.

Procedure:

To create a table in a partitioned database environment using the command line, enter:

```
CREATE TABLE <name>
  (<column_name> <data_type> <null_attribute>)
  IN <tagle_space_name>
  INDEX IN <index_space_name>
  LONG IN <long_space_name>
  DISTRIBUTE BY HASH (<column_name>)
```

Following is an example:

```
CREATE TABLE MIXREC (MIX_CNTL INTEGER NOT NULL,
                     MIX_DESC CHAR(20) NOT NULL,
                     MIX_CHR  CHAR(9) NOT NULL,
                     MIX_INT INTEGER NOT NULL,
                     MIX_INTS SMALLINT NOT NULL,
                     MIX_DEC DECIMAL NOT NULL,
                     MIX_FLT FLOAT NOT NULL,
                     MIX_DATE DATE NOT NULL,
                     MIX_TIME TIME NOT NULL,
                     MIX_TMSTMP TIMESTAMP NOT NULL)
                     IN MIXTS12
                     DISTRIBUTE BY HASH (MIX_INT)
```

In the preceding example, the table space is `MIXTS12` and the distribution key is `MIX_INT`. If the distribution key is not specified explicitly, it is `MIX_CNTL`. (If no primary key is specified and no distribution key is defined, the distribution key is the first non-long column in the list.)

A row of a table, and all information about that row, always resides on the same database partition.

Related concepts:
- "Database partition group design" on page 251
- "Database partition groups," on page 250
- "Table collocation" on page 255

Related reference:
- "CREATE TABLE " on page 956

Creating a view

Views are derived from one or more base tables, nicknames, or views, and can be used interchangeably with base tables when retrieving data. When changes are made to the data shown in a view, the data is changed in the table itself.

A view can be created to limit access to sensitive data, while allowing more general access to other data.

When inserting into a view where the SELECT-list of the view definition directly or indirectly includes the name of an identity column of a base table, the same rules apply as if the INSERT statement directly referenced the identity column of the base table.

In addition to using views as described above, a view can also be used to:
- Alter a table without affecting application programs. This can happen by creating a view based on an underlying table. Applications that use the underlying table are not affected by the creation of the new view. New applications can use the created view for different purposes than those applications that use the underlying table.
- Sum the values in a column, select the maximum values, or average the values.
- Provide access to information in one or more data sources. You can reference nicknames within the CREATE VIEW statement and create multi-location/global views (the view could join information in multiple data sources located on different systems).

  When you create a view that references nicknames using standard CREATE VIEW syntax, you will see a warning alerting you to the fact that the authentication ID of view users will be used to access the underlying object or objects at data sources instead of the view creator authentication ID. Use the FEDERATED keyword to suppress this warning.

A typed view is based on a predefined structured type. You can create a typed view using the CREATE VIEW statement.

An alternative to creating a view is to use a nested or common table expression to reduce catalog lookup and improve performance.

Prerequisites:

The base table, nickname, or view on which the view is to be based must already exist before the view can be created.

Restrictions:

You can create a view that uses a UDF in its definition. However, to update this view so that it contains the latest functions, you must drop it and then re-create it. If a view is dependent on a UDF, that function cannot be dropped.

The following SQL statement creates a view with a function in its definition:

```
CREATE VIEW EMPLOYEE_PENSION (NAME, PENSION)
  AS SELECT NAME, PENSION(HIREDATE,BIRTHDATE,SALARY,BONUS)
  FROM EMPLOYEE
```

The UDF function PENSION calculates the current pension an employee is eligible to receive, based on a formula involving their HIREDATE, BIRTHDATE, SALARY, and BONUS.

Procedure:

To create a view using the Control Center:

1. Expand the object tree until you see the **Views** folder.
2. Right-click the **Views** folder, and select **Create** from the pop-up menu.
3. Complete the information, and click **OK**.

To create a view using the command line, enter:

```
CREATE VIEW <name> (<column>, <column>, <column>)
   SELECT <column_names> FROM <table_name>
   WITH CHECK OPTION
```

For example, the EMPLOYEE table might have salary information in it, which should not be made available to everyone. The employee’s phone number, however, should be generally accessible. In this case, a view could be created from the LASTNAME and PHONENO columns only. Access to the view could be granted to PUBLIC, while access to the entire EMPLOYEE table could be restricted to those who have the authorization to see salary information.

With a view, you can make a subset of table data available to an application program and validate data that is to be inserted or updated. A view can have column names that are different from the names of corresponding columns in the original tables.

The use of views provides flexibility in the way your programs and end-user queries can look at the table data.

The following SQL statement creates a view on the EMPLOYEE table that lists all employees in Department A00 with their employee and telephone numbers:

```
CREATE VIEW EMP_VIEW (DA00NAME, DA00NUM, PHONENO)
   AS SELECT LASTNAME, EMPNO, PHONENO FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

The first line of this statement names the view and defines its columns. The name EMP_VIEW must be unique within its schema in SYSCAT.TABLES. The view name appears as a table name although it contains no data. The view will have three columns called DA00NAME, DA00NUM, and PHONENO, which correspond to the columns LASTNAME, EMPNO, and PHONENO from the EMPLOYEE table.

The column names listed apply one-to-one to the select list of the SELECT statement. If column names are not specified, the view uses the same names as the columns of the result table of the SELECT statement.

The second line is a SELECT statement that describes which values are to be selected from the database. It might include the clauses ALL, DISTINCT, FROM, WHERE, GROUP BY, and HAVING. The name or names of the data objects from which to select columns for the view must follow the FROM clause.

The WITH CHECK OPTION clause indicates that any updated or inserted row to the view must be checked against the view definition, and rejected if it does not conform. This enhances data integrity but requires additional processing. If this clause is omitted, inserts and updates are not checked against the view definition.

The following SQL statement creates the same view on the EMPLOYEE table using the SELECT AS clause:

```
CREATE VIEW EMP_VIEW
   SELECT LASTNAME AS DA00NAME,
          EMPNO AS DA00NUM,
          PHONENO
   FROM EMPLOYEE
   WHERE WORKDEPT = 'A00'
   WITH CHECK OPTION
```

Related concepts:
- “Views” in *SQL Reference, Volume 1*
- “Controlling access to data with views” on page 101
- “Table and view privileges” on page 79
- “Updating view contents using triggers” in *Administration Guide: Implementation*

Related tasks:
- “Altering or dropping a view” in *Administration Guide: Implementation*
- “Recovering inoperative views” in *Administration Guide: Implementation*
- “Removing rows from a table or view” in *Administration Guide: Implementation*
- “Creating typed views” in *Developing SQL and External Routines*

Related reference:
- “CREATE VIEW ” on page 1034
- “INSERT ” on page 1109

Creating an index

An *index* is a set of one or more keys, each pointing to rows in a table. An index allows more efficient access to rows in a table by creating a direct path to the data through pointers.

Procedure:

Performance Tip: If you are going to carry out the following series of tasks:
1. Create Table
2. Load Table
3. Create Index (without the COLLECT STATISTICS option)
4. Perform RUNSTATS

Or, if you are going to carry out the following series of tasks:
1. Create Table
2. Load Table
3. Create Index (with the COLLECT STATISTICS option)

then you should consider ordering the execution of tasks in the following way:
1. Create the table
2. Create the index
3. Load the table with the `statistics yes` option requested.

Indexes are maintained after they are created. Subsequently, when application programs use a key value to randomly access and process rows in a table, the index based on that key value can be used to access rows directly. This is important, because the physical storage of rows in a base table is not ordered.

When creating a multi-dimensional clustering (MDC) table, block indexes are created. Regular indexes point to individual rows; block indexes point to blocks or extents of data, and are much smaller than regular indexes. For a partitioned MDC table, the MDC block indexes are stored in the default table space or the table space defined in the INDEXES IN clause of CREATE TABLE statement.

In partitioned database environments, where table data is distributed across database partitions, the index object is distributed across the database partitions in the same manner as the table. For regular tables this would mean one index object per database partition. When creating partitioned tables, non-partitioned indexes are created. Each index is an independent object shared among all the data partitions of the table and stored in a single table space.

Table 69. Index types and locations.

Table type	Index description	Index storage location
MDC	Block	By default, the index is stored in same table space as the table data, however, you can specify different table space using the `INDEX IN` clause on the CREATE TABLE statement.
Distributed	Independent (not shared)	The index is distributed across the database partitions.
Partitioned	Non-partitioned (shared)	The index is stored in a single table space.
Partitioned	Partitioned (shared)	The index is stored in a single table space in a separate index object. You can also specify a different table space for each index on the table.

When a row is inserted, unless there is a clustering index defined, the row is placed in the most convenient storage location that can accommodate it. When searching for rows of a table that meet a particular selection condition and the table has no indexes, the entire table is scanned. An index optimizes data retrieval without performing a lengthy sequential search.

The data for your indexes can be stored in the same table space as your table data, or in a separate table space containing index data. The table space used to store the index data is determined when the table is created, or for partitioned tables, the index location can be overridden using the `IN` clause of the CREATE INDEX statement. This allows different table spaces to be specified for different indexes, as required.

For example, to create an index on a partitioned table, assume there is already a partitioned table foo (a int, b int, c int). To create a unique index, a_idx in the table space my_tbsp, use this command:

```
CREATE UNIQUE INDEX a_idx ON foo ( a ) IN my_tbsp
```

To create an index using the Control Center:

1. Expand the object tree until you see the **Indexes** folder.
2. Right-click the **Indexes** folder, and select **Create —> Index Using Wizard** from the pop-up menu.
3. Follow the steps in the wizard to complete your task.

To create an index using the command line, enter:

```
   CREATE INDEX <name> ON <table_name> (<column_name>)
```

Related concepts:
- "Optimizing load performance" in *Data Movement Utilities Guide and Reference*
- "Understanding index behavior on partitioned tables" in *Performance Guide*
- "Index cleanup and maintenance" in *Performance Guide*
- "Relational index performance tips" in *Performance Guide*
- "Relational index planning tips" in *Performance Guide*
- "Index privileges" on page 82
- "Options on the CREATE INDEX statement" on page 316
- "Using an index" on page 315

Related tasks:
- "Dropping an index, index extension, or an index specification" in *Administration Guide: Implementation*
- "Renaming an existing table or index" in *Administration Guide: Implementation*

Related reference:
- "CREATE INDEX " on page 921
- "Restrictions on native XML data store" in *XML Guide*

Using an index

An index is never directly used by an application program. The decision on whether to use an index and which of the potentially available indexes to use is the responsibility of the optimizer.

The best index on a table is one that:
- Uses high-speed disks
- Is highly-clustered
- Is made up of only a few narrow columns
- Uses columns with high cardinality

Related concepts:
- "Data access through index scans" in *Performance Guide*
- "Relational index planning tips" in *Performance Guide*

- “Relational index performance tips” in *Performance Guide*
- “Table and index management for MDC tables” in *Performance Guide*
- “Table and index management for standard tables” in *Performance Guide*

Options on the CREATE INDEX statement

You can create an index that will allow duplicates (a non-unique index) to enable efficient retrieval by columns other than the primary key, and allow duplicate values to exist in the indexed column or columns.

The following SQL statement creates a non-unique index called LNAME from the LASTNAME column on the EMPLOYEE table, sorted in ascending order:

```
CREATE INDEX LNAME ON EMPLOYEE (LASTNAME ASC)
```

The following SQL statement creates a unique index on the phone number column:

```
CREATE UNIQUE INDEX PH ON EMPLOYEE (PHONENO DESC)
```

A unique index ensures that no duplicate values exist in the indexed column or columns. The constraint is enforced at the end of the SQL statement that updates rows or inserts new rows. This type of index cannot be created if the set of one or more columns already has duplicate values.

The keyword ASC puts the index entries in ascending order by column, while DESC puts them in descending order by column. The default is ascending order.

You can create a unique index on two columns, one of which is an include column. The primary key is defined on the column that is not the include column. Both of them are shown in the catalog as primary keys on the same table. Normally there is only one primary key per table.

The INCLUDE clause specifies additional columns to be appended to the set of index key columns. Any columns included with this clause are not used to enforce uniqueness. The included columns might improve the performance of some queries through index-only access. The columns must be distinct from the columns used to enforce uniqueness (otherwise you will receive error message SQLSTATE 42711). The limits for the number of columns and sum of the length attributes apply to all of the columns in the unique key and in the index.

A check is performed to determine if an existing index matches the primary key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (either ascending or descending) specifications. If a matching index definition is found, the description of the index is changed to indicate that it is the primary index, as required by the system, and it is changed to unique (after ensuring uniqueness) if it was a non-unique index.

This is why it is possible to have more than one primary key on the same table as indicated in the catalog.

When working with a structured type, it might be necessary to create user-defined index types. This requires a means of defining index maintenance, index search, and index exploitation functions.

The following SQL statement creates a clustering index called INDEX1 on the LASTNAME column of the EMPLOYEE table:

```
CREATE INDEX INDEX1 ON EMPLOYEE (LASTNAME) CLUSTER
```

To use the internal storage of the database effectively, use clustering indexes with the PCTFREE parameter associated with the ALTER TABLE statement so that new data can be inserted on the correct pages. When data is inserted on the correct pages, clustering order is maintained. Typically, the greater the INSERT activity on the table, the larger the PCTFREE value (on the table) that will be needed in order to maintain clustering. Since this index determines the order by which the data is laid out on physical pages, only one clustering index can be defined for any particular table.

If the index key values of these new rows are always new high key values for example, then the clustering attribute of the table will try to place them at the end of the table. Having free space in other pages will do little to preserve clustering. In this case, placing the table in append mode might be a better choice than a clustering index and altering the table to have a large PCTFREE value. You can place the table in append mode by issuing: ALTER TABLE APPEND ON.

The above discussion also applies to new "overflow" rows that result from UPDATEs that increase the size of a row.

A single index created using the ALLOW REVERSE SCANS parameter on the CREATE INDEX statement can be scanned in a forward or a backward direction. That is, such indexes support scans in the direction defined when the index was created and scans in the opposite or reverse direction. The statement could look something like:

```
   CREATE INDEX iname ON tname (cname DESC) ALLOW REVERSE SCANS
```

In this case, the index (`iname`) is formed based on descending values (`DESC`) in the given column (`cname`). By allowing reverse scans, although the index on the column is defined for scans in descending order, a scan can be done in ascending order (reverse order). The actual use of the index in both directions is not controlled by you but by the optimizer when creating and considering access plans.

The MINPCTUSED clause of the CREATE INDEX statement specifies the threshold for the minimum amount of used space on an index leaf page. If this clause is used, online index defragmentation is enabled for this index. Once enabled, the following considerations are used to determine if an online index defragmentation takes place: After a key is physically removed from a leaf page of this index and a percentage of used space on the page is less than the specified threshold value, the neighboring index leaf pages are checked to determine if the keys on the two leaf pages can be merged into a single index leaf page.

For example, the following SQL statement creates an index with online index defragmentation enabled:

```
   CREATE INDEX LASTN ON EMPLOYEE (LASTNAME) MINPCTUSED 20
```

When a key is physically removed from an index page of this index, if the remaining keys on the index page take up twenty percent or less space on the index page, then an attempt is made to delete an index page by merging the keys of this index page with those of a neighboring index page. If the combined keys can all fit on a single page, this merge is performed and one of the index pages is deleted.

The CREATE INDEX statement allows you to create the index while, at the same time, allowing read and write access to the underlying table and any previously existing indexes. To restrict access to the table while creating the index, use the LOCK TABLE statement to lock the table before creating the index. The new index is created by scanning the underlying table. Any changes made to the table while the index is being created are logged. Once the new index is created, the changes are applied to the index. To apply the logged changes more quickly during the index creation, a separate copy of the changes is maintained in memory buffer space, which is allocated on demand from the utility heap. This allows the index creation to process the changes by directly reading from memory first, and reading through the logs, if necessary, at a much later time. Once all the changes have been applied to the index, the table is quiesced while the new index is made visible.

When creating a unique index, ensure that there are no duplicate keys in the table and that the concurrent inserts during index creation are not going to introduce duplicate keys. Index creation uses a deferred unique scheme to detect duplicate keys, and therefore no duplicate keys will be detected until the very end of index creation, at which point the index creation will fail because of the duplicate keys.

The PCTFREE clause of the CREATE INDEX statement specifies the percentage of each index page to leave as free space when the index is built. Leaving more free space on the index pages will result in fewer page splits. This will reduce the need to reorganize the table in order to regain sequential index pages which increases prefetching. And prefetching is one important component that might improve performance. Again, if there are always high key values, then you will want to consider lowering the value of the PCTFREE clause of the CREATE INDEX statement. In this way there will be limited wasted space reserved on each index page.

The LEVEL2 PCTFREE clause directs the system to preserve a specified percentage of free space on each page in the second level of an index. You specify a percentage of free space when the index is created to accommodate future insertions and updates. The second level is the level immediately above the leaf level. The default is to preserve a minimum of 10 and the PCTFREE value in all non-leaf pages. The LEVEL2 PCTFREE parameter allows the default to be overwritten; if you use the LEVEL2 PCTFREE integer option in the CREATE INDEX statement, the integer percent of free space is left on level 2 intermediate pages. A minimum of 10 and the integer percent of free space is left on level 3 and higher intermediate pages. By leaving more free space on the second level, the number of page splits that occur at the second level of the index is reduced.

The PAGE SPLIT SYMMETRIC, PAGE SPLIT HIGH, and PAGE SPLIT LOW clauses allow a choice in the page split behavior when inserting into an index.

The PAGE SPLIT SYMMETRIC clause is a default page split behavior that splits roughly in the middle of an index page. Using this default behavior is best when the insertion into an index is random or does not follow one of the patterns that are addressed by the PAGE SPLIT HIGH and PAGE SPLIT LOW clauses.

The PAGE SPLIT HIGH behavior is useful when there are ever increasing ranges in the index. Increasing ranges in the index might occur when:

- There is an index with multiple key parts and there are many values (multiple index pages worth) where all except the last key part have the same value
- All inserts into the table would consist of a new value which has the same value as existing keys for all but the last key part
- The last key part of the inserted value is larger than that of the existing keys

For example, if an index has the following key values:

```
(1,1),(1,2),(1,3), ... (1,n),
(2,1),(2,2),(2,3), ... (2,n),
...
(m,1),(m,2),(m,3), ...(m,n)
```

then the next key to be inserted would have the value (x,y) where 1 <= x <= m and y > n. If the insertions follow such a pattern, the PAGE SPLIT HIGH clause can be used so that page splits do not result in many pages that are fifty percent empty.

Similarly, PAGE SPLIT LOW can be used when there are ever-decreasing ranges in the index, to avoid leaving pages 50 percent empty.

Note: If you want to add a primary or unique key, and you want the underlying index to use SPLIT HIGH, SPLIT LOW, PCTFREE, LEVEL2 PCTFREE, MINPCTUSED, CLUSTER, or ALLOW REVERSE SCANS you must first create an index specifying the desired keys and parameters. Then use an ALTER TABLE statement to add the primary or unique key. The ALTER TABLE statement will pick up and reuse the index that you have already created.

You can collect index statistics as part of the creation of the index. At the time when you use the CREATE INDEX statement, the key value statistics and the physical statistics are available for use. By collecting the index statistics as part of the CREATE INDEX statement, you will not need to run the RUNSTATS utility immediately following the completion of the CREATE INDEX statement.

For example, the following SQL statement will collect basic index statistics as part of the creation of an index:

```
CREATE INDEX IDX1 ON TABL1 (COL1) COLLECT STATISTICS
```

If you have a replicated summary table, its base table (or tables) must have a unique index, and the index key columns must be used in the query that defines the replicated summary table.

For intra-partition parallelism, create index performance is improved by using multiple processors for the scanning and sorting of data that is performed during index creation. The use of multiple processors is enabled by setting *intra_parallel* to YES(1) or SYSTEM(-1). The number of processors used during index creation is determined by the system and is not affected by the configuration parameters *dft_degree* or *max_querydegree*, by the application runtime degree, or by the query compilation degree.

In multiple partition databases, unique indexes must be defined as supersets of the distribution key.

Related concepts:

- "Index reorganization" in *Performance Guide*

- "Online index defragmentation" in *Performance Guide*
- "Relational index performance tips" in *Performance Guide*
- "Table and index management for MDC tables" in *Performance Guide*
- "Table and index management for standard tables" in *Performance Guide*

Related tasks:
- "Changing table attributes" in *Administration Guide: Implementation*

Related reference:
- "CREATE INDEX " on page 921
- "dft_degree - Default degree configuration parameter" in *Performance Guide*
- "intra_parallel - Enable intra-partition parallelism configuration parameter" in *Performance Guide*
- "max_querydegree - Maximum query degree of parallelism configuration parameter" in *Performance Guide*

Chapter 25. Configuring DB2 to be Common Criteria compliant

A Common Criteria compliant DB2 instance must be configured so that:

- Any request made by a DB2 client to a DB2 server is authenticated by the server before being processed, and
- Communications occur using TCP/IP

The following topic provides the steps required to configure your environment so that it is Common Criteria compliant.

Configuring DB2 to be Common Criteria compliant

Immediately *after* installing DB2, modify the values of the *authentication* and *svcename* database manager configuration parameters. Changing the values of these configuration parameters will ensure that your environment conforms to the Common Criteria requirements.

Prerequisites:

To update configuration parameter values, you require the SYSADM authority level.

Procedure:

1. Update the database manager configuration so that clients must authenticate themselves at the DB2 server via a user ID and password. Issue the following command:
   ```
   db2 update dbm cfg using authentication server
   ```
2. Update the database manager configuration to specify the TCP/IP port that DB2 will use to await communications with remote client nodes. Issue the following command:
   ```
   db2 update dbm cfg using svcename port-number
   ```
   The port number that you specify must be the first of two consecutive ports that are reserved for use by DB2. For additional information about the *svcename* database manager configuration parameter, see ″svcename - TCP/IP service name″
3. Stop DB2. Issue the following command:
   ```
   db2stop
   ```
4. Start DB2. Issue the following command:
   ```
   db2start
   ```

The updated database manager configuration parameters take effect when DB2 is restarted.

Related concepts:

- “System administration authority (SYSADM)” on page 71

Related reference:

- “ATTACH ” on page 631

- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629
- “authentication - Authentication type ” on page 1497

Part 7. Altering a partitioned database environment

Chapter 26. Altering a database partition group

Procedure:

To alter a database partition group using the Control Center:

1. Open the Alter Database Partition Group wizard. To open the Alter Database Partition Group wizard: From the Control Center, expand the object tree until you find the **Database Partition Groups** folder. Click the **Database Partition Groups** folder. Any existing database partition groups are displayed in the contents pane on the right. Right-click the database partition group you want to change and select **Alter** from the pop-up menu. The Alter Database Partition Group wizard opens.

   You can also open the Alter Database Partition Group window from the Storage Management view. To open the Storage Management view: From the Control Center window, expand the object tree until you find the database, database partition group, or table space you want to examine in the Storage Management view. Right-click the desired database and select **Manage Storage** from the pop-up menu. The Storage Management view opens.
   Note: The first time you launch the Storage Management view from an object you will need to specify your settings in the Storage Management Setup launchpad.
2. Complete each of the applicable wizard pages. The **Finish** push button is enabled when you complete enough information for the wizard to alter the database partition group.

To alter a database partition group using the command line processor: use the `REDISTRIBUTE DATABASE PARTITION GROUP` command.

Once you add or drop database partitions, you must redistribute the current data across the new set of database partitions in the database partition group.

Related concepts:
- "Data redistribution" on page 337
- "Management of database server capacity" on page 327

Related tasks:
- "Redistributing data across database partitions" on page 339

Related reference:
- "REDISTRIBUTE DATABASE PARTITION GROUP " on page 577

Chapter 27. Scaling your configuration

This chapter describes how you can manage database capacity, primarily by adding and dropping database partitions. Other methods of increasing capacity include adding CPUs and adding memory.

Management of database server capacity

If database manager capacity does not meet your present or future needs, you can expand its capacity in the following ways:

- Add disk space and create additional containers.
- Add memory.

If these simple strategies do not add the capacity you need, consider the following methods:

- Add processors.

  If a single-partition database configuration with a single processor is used to its maximum capacity, you might either add processors or add database partitions. The advantage of adding processors is greater processing power. In an SMP system, processors share memory and storage system resources. All of the processors are in one system, so there are no additional overhead considerations such as communication between systems and coordination of tasks between systems. Utilities in DB2 such as load, backup, and restore can take advantage of the additional processors. DB2 database supports this environment.

  Note: Some operating systems, such as the Solaris operating system, can dynamically turn processors on- and off-line.

  If you add processors, review and modify some database configuration parameters that determine the number of processors used. The following database configuration parameters determine the number of processors used and might need to be updated:

  – Default degree (dft_degree)
  – Maximum degree of parallelism (max_querydegree)
  – Enable intra-partition parallelism (intra_parallel)

  You should also evaluate parameters that determine how applications perform parallel processing.

  In an environment where TCP/IP is used for communication, review the value for the DB2TCPCONNMGRS registry variable.

- Add physical partitions.

  If your database manager is currently in a partitioned database environment, you can increase both data-storage space and processing power by adding separate single-processor or multiple-processor physical partitions. The memory and storage system resources on each database partition are not shared with the other database partitions. Although adding database partitions might result in communication and task-coordination issues, this choice provides the advantage of balancing data and user access across more than one system. DB2 database supports this environment.

You can add database partitions either while the database manager system is running or while it is stopped. If you add database partitions while the system is running, however, you must stop and restart the system before databases migrate to the new database partition.

When you scale your system by changing the environment, you should be aware of the impact that such a change can have on your database procedures such as loading data, backing up the database, and restoring the database.

When you add a new database partition, you cannot drop or create a database that takes advantage of the new database partition until the procedure is complete, and the new server is successfully integrated into the system.

Related concepts:
- "Adding database partitions in a partitioned database environment" on page 328

Adding database partitions in a partitioned database environment

You can add database partitions to the partitioned database system either when it is running, or when it is stopped. Because adding a new server can be time consuming, you may want to do it when the database manager is already running.

Use the ADD DBPARTITIONNUM command to add a database partition to a system. This command can be invoked in the following ways:
- As an option on `db2start`
- With the command-line processor ADD DBPARTITIONNUM command
- With the API function `sqleaddn`
- With the API function `sqlepstart`

If your system is stopped, you use `db2start`. If it is running, you can use any of the other choices.

When you use the ADD DBPARTITIONNUM command to add a new database partition to the system, all existing databases in the instance are expanded to the new database partition. You can also specify which containers to use for temporary table spaces for the databases. The containers can be:
- The same as those defined for the catalog partition for each database. (This is the default.)
- The same as those defined for another database partition.
- Not created at all. You must use the ALTER TABLESPACE statement to add temporary table space containers to each database before the database can be used.

You cannot use a database on the new database partition to contain data until one or more database partition groups are altered to include the new database partition.

You cannot change from a single-partition database to a multi-partition database by simply adding a database partition to your system. This is because the redistribution of data across database partitions requires a distribution key on each affected table. The distribution keys are automatically generated when a table is created in a multi-partition database. In a single-partition database, distribution keys can be explicitly created with the CREATE TABLE or ALTER TABLE SQL statements.

Note: If no databases are defined in the system and you are running Enterprise Server Edition on a UNIX operating system, edit the `db2nodes.cfg` file to add a new database partition definition; do not use any of the procedures described, as they apply only when a database exists.

Windows Considerations: If you are using Enterprise Server Edition on Windows and have no databases in the instance, use the DB2NCRT command to scale the database system. If, however, you already have databases, use the DB2START ADDNODE command to ensure that a database partition is created for each existing database when you scale the system. On Windows, you should never manually edit the node configuration file (db2nodes.cfg), as this can introduce inconsistencies into the file.

Related tasks:
- "Adding a database partition to a running database system" on page 329
- "Adding a database partition to a stopped database system on Windows" on page 330
- "Dropping a database partition" on page 335

Adding a database partition to a running database system

You can add new database partitions to a partitioned database environment while it is running and while applications are connected to databases. However, a newly added server does not become available to all databases until the database manager is shut down and restarted.

Procedure:

To add a database partition to a running database manager using the Control Center:

1. Open the Add Partitions wizard:
   a. From the Control Center, expand the object tree until you find the instance object that you want to work with. Right-click the object, and click **Add Partitions** from the pop-up menu. The Add Partitions launchpad opens.
   b. Click the **Add Partitions** button. The Add Partitions wizard opens.
2. Complete each of the applicable wizard pages. Click the wizard overview link on the first page for more information. The **Finish** push button is available when you complete enough information for the wizard to add the partition.

To add a database partition to a running database manager using the command line:

1. On any existing database partition, run the DB2START command.

   On all platforms, specify the new database partition values for DBPARTITIONNUM, ADD DBPARTITIONNUM, HOSTNAME, PORT, and NETNAME parameters. On the Windows platform, you also specify the COMPUTER, USER, and PASSWORD parameters.

   You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

When the DB2START command is complete, the new server is stopped.

2. Stop the database manager on all database partitions by running the DB2STOP command.

   When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDNODE parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
3. Start the database manager by running the DB2START command.

   The newly added database partition is now started along with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
4. Back up all databases on the new database partition. (Optional)
5. Redistribute data to the new database partition. (Optional)

Related concepts:

- "Adding database partitions in a partitioned database environment" on page 328

Related tasks:

- "Adding a database partition to a stopped database system on UNIX" on page 332
- "Adding a database partition to a stopped database system on Windows" on page 330

Adding a database partition to a stopped database system on Windows

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started up again.

Prerequisites:

You must install the new server before you can create a database partition on it.

Procedure:

To add a database partition to a stopped partitioned database server using the Control Center:

1. Open the Add Partitions wizard:
   a. From the Control Center, expand the object tree until you find the instance object that you want to work with. Right-click the object, and click **Add Partitions** from the pop-up menu. The Add Partitions launchpad opens.
   b. Click the **Add Partitions** button. The Add Partitions wizard opens.
2. Complete each of the applicable wizard pages. Click the wizard overview link on the first page for more information. The **Finish** push button is available when you complete enough information for the wizard to add the partition.

To add a database partition to a stopped partitioned database server using the comand line:

1. Issue DB2STOP to stop all the database partitions.
2. Run the ADD DBPARTITIONNUM command on the new server.

   A database partition is created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
3. Run the DB2START command to start the database system. Note that the node configuration file (cfg) has already been updated to include the new server during the installation of the new server.
4. Update the configuration file on the new database partition as follows:
   a. On any existing database partitions, run the DB2START command.

      Specify the new database partition values for DBPARTITIONNUM, ADDDBPARTITIONNUM, HOSTNAME, PORT, and NETNAME parameters as well as the COMPUTER, USER, and PASSWORD parameters.

      You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

      When the DB2START command is complete, the new server is stopped.
   b. Stop the entire database manager by running the DB2STOP command.

      When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDDBPARTITIONNUM parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
5. Start the database manager by running the DB2START command.

   The newly added database partition is now started with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Back up all databases on the new database partition. (Optional)
7. Redistribute data to the new database partition. (Optional)

Related concepts:

- “Error recovery when adding database partitions” on page 334
- “Adding database partitions in a partitioned database environment” on page 328

Related tasks:

- “Adding a database partition to a running database system” on page 329
- “Adding a database partition to a stopped database system on UNIX” on page 332

Adding a database partition to a stopped database system on UNIX

You can add new database partitions to a partitioned database system while it is stopped. The newly added database partition becomes available to all databases when the database manager is started up again.

Prerequisites:

You must install the new server if it does not exist, including the following tasks:

- Making executables accessible (using shared file-system mounts or local copies)
- Synchronizing operating system files with those on existing processors
- Ensuring that the `sqllib` directory is accessible as a shared file system
- Ensuring that the relevant operating system parameters (such as the maximum number of processes) are set to the appropriate values

You must also register the host name with the name server or in the hosts file in the etc directory on all database partitions.

Procedure:

To add a database partition to a stopped partitioned database server using the Control Center:

1. Open the Add Partitions wizard:
   a. From the Control Center, expand the object tree until you find the instance object that you want to work with. Right-click the object, and click **Add Partitions** from the pop-up menu. The Add Partitions launchpad opens.
   b. Click the **Add Partitions** button. The Add Partitions wizard opens.
2. Complete each of the applicable wizard pages. Click the wizard overview link on the first page for more information. The **Finish** push button is available when you complete enough information for the wizard to add the partition.

To add a database partition to a stopped partitioned database server using the command line:

1. Issue DB2STOP to stop all the database partitions.
2. Run the ADD DBPARTITIONNUM command on the new server.

   A database partition is created locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.

3. Run the DB2START command to start the database system. Note that the node configuration file (cfg) has already been updated to include the new server during the installation of the new server.
4. Update the configuration file on the new database partition as follows:
   a. On any existing database partition, run the DB2START command.

      Specify the new database partition values for DBPARTITIONNUM, ADDDBPARTITIONNUM, HOSTNAME, PORT, and NETNAME parameters as well as the COMPUTER, USER, and PASSWORD parameters.

      You can also specify the source for any temporary table space container definitions that need to be created with the databases. If you do not provide table space information, temporary table space container definitions are retrieved from the catalog partition for each database.

      When the DB2START command is complete, the new server is stopped.
   b. Stop the entire database manager by running the DB2STOP command.

      When you stop all the database partitions in the system, the node configuration file is updated to include the new database partition. The node configuration file is not updated with the new server information until DB2STOP is executed. This ensures that the ADD DBPARTITIONNUM command, which is called when you specify the ADDDBPARTITIONNUM parameter to the DB2START command, runs on the correct database partition. When the utility ends, the new server partition is stopped.
5. Start the database manager by running the DB2START command.

   The newly added database partition is now started with the rest of the system.

   When all the database partitions in the system are running, you can run system-wide activities, such as creating or dropping a database.

   Note: You might have to issue the DB2START command twice for all database partition servers to access the new `db2nodes.cfg` file.
6. Back up all databases on the new database partition. (Optional)
7. Redistribute data to the new database partition. (Optional)

You can also update the configuration file manually, as follows:
1. Edit the db2nodes.cfg file and add the new database partition to it.
2. Issue the following command to start the new database partition: `DB2START DBPARTITIONNUM partitionnum`

   Specify the number you are assigning to the new database partition as the value of nodenum.
3. If the new server is to be a logical partition (that is, it is not database partition 0), use db2set command to update the DBPARTITIONNUM registry variable. Specify the number of the database partition you are adding.
4. Run the ADD NODE command on the new database partition.

   This command creates a database partition locally for every database that already exists in the system. The database parameters for the new database partitions are set to the default value, and each database partition remains empty until you move data to it. Update the database configuration parameter values to match those on the other database partitions.
5. When the ADD DBPARTITIONNUM command completes, issue the DB2START command to start the other database partitions in the system.

   Do not perform any system-wide activities, such as creating or dropping a database, until all database partitions are successfully started.

Related concepts:

- "Error recovery when adding database partitions" on page 334

Related tasks:

- "Adding a database partition to a running database system" on page 329
- "Adding a database partition to a stopped database system on Windows" on page 330
- "Dropping a database partition" on page 335

Error recovery when adding database partitions

In version 8.1 and later, adding database partitions does not fail as a result of of non-existent buffer pools because DB2 creates system buffer pools to provide default automatic support for all buffer-pool page sizes. However, if one of these system buffer pools is used, performance might be seriously affected because the system buffer pools are very small. If a system buffer pool is used, a message is written to the administration notification log.

System buffer pools are used in database partition addition scenarios in the following circumstances:

- You add database partitions to a partitioned database environment that has one or more system temporary table spaces with a page size that is different from the default of 4 KB. When a database partition is created, only the IBMDEFAULTDP buffer pool exists, and this buffer pool has a page size of 4 KB.

  Consider the following examples:

  1. You use the db2start command to add a database partition to the current multi-partition database:

     ```
     DB2START DBPARTITIONNUM 2 ADD DBPARTITIONNUM HOSTNAME newhost PORT 2
     ```
  2. You use the ADD DBPARTITIONNUM command after you manually update the `db2nodes.cfg` file with the new database partition description.

  One way to prevent these problems is to specify the WITHOUT TABLESPACES clause on the ADD NODE or the **db2start** command. After doing this, you need to use the CREATE BUFFERPOOL statement to create the buffer pools using , and associate the system temporary table spaces to the buffer pool using the ALTER TABLESPACE statement.
- You add database partitions to an existing database partition group that has one or more table spaces with a page size that is different from the default page size, which is 4 KB. This occurs because the non-default page-size buffer pools created on the new database partition have not been activated for the table spaces.

  Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

  Consider the following example:

  – You use the ALTER DATABASE PARTITION GROUP statement to add a database partition to a database partition group, as follows:

    ```
    DB2START
    CONNECT TO mpp1
    ALTER DATABASE PARTITION GROUP ng1 ADD NODE (2)
    ```

    One way to prevent this problem is to create buffer pools for each page size and then to reconnect to the database before issuing the following ALTER DATABASE PARTITION GROUP statement:

```
DB2START
CONNECT TO mpp1
CREATE BUFFERPOOL bp1 SIZE 1000 PAGESIZE 8192
CONNECT RESET
CONNECT TO mpp1
ALTER DATABASE PARTITION GROUP ng1 ADD NODE (2)
```

Note: If the database partition group has table spaces with the default page size, message SQL1759W is returned:

Related tasks:

- "Adding a database partition to a running database system" on page 329
- "Adding a database partition to a stopped database system on Windows" on page 330

Dropping a database partition

You can drop a database partition that is not being used by any database and free the computer for other uses.

Prerequisites:

Verify that the database partition is not in use by issuing the DROP DBPARTITION VERIFY command or the *sqledrpn* API.

- If you receive message SQL6034W (Database partition not used in any database), you can drop the database partition.
- If you receive message SQL6035W (Database partition in use by database), use the `REDISTRIBUTE NODEGROUP` command to redistribute the data from the database partition that you are dropping to other database partitions from the database alias.

Also ensure that all transactions for which this database partition was the coordinator have all committed or rolled back successfully. This might require doing crash recovery on other servers. For example, if you drop the coordinator partition, and another database partition participating in a transaction crashed before the coordinator partition was dropped, the crashed database partition will not be able to query the coordinator partition for the outcome of any in-doubt transactions.

Procedure:

To drop a database partition using the Control Center:

1. Optional: Back up the database.
2. Open the Drop Database Partitions launchpad. To open the Drop Database Partitions launchpad:
   a. Open the Database Partitions view. To open the Database Partitions wiew: From the Control Center, expand the object tree until you find the instance for which you want to view the database partitions. Right-click on the instance you want and select **Open–>Database Partitions** from the pop-up menu. The Database Partitions wiew opens for the selected instance.
   b. Select the database partitions you want to drop.
   c. Right-click the selected database partitions and click **Drop** in the pop-up menu. The Drop Database Partitions launchpad opens.
3. Drop the database partitions from database partition groups.
   Note: This operation does not drop the database partitions immediately. Instead, it flags the database partitions that you want to drop so that data can be move off them when you redistribute the data in the database partition group.

To drop a database partition using the command line processor:

1. Issue the `DB2STOP` command with the DROP NODENUM parameter to drop the database partition. After the command completes successfully, the system is stopped.
2. Start the database manager with the `DB2START` command.

Related concepts:

- "Management of database server capacity" on page 327
- "Adding database partitions in a partitioned database environment" on page 328

Related reference:

- "DROP DBPARTITIONNUM VERIFY " on page 495
- "sqledrpn - Check whether a database partition server can be dropped" on page 810

Chapter 28. Redistributing Data Across Database Partitions

This chapter provides information about determining when to redistribute data across database partitions, how to perform the redistribution, and how to recover from redistribution errors.

Data redistribution

To redistribute table data among the database partitions in a partitioned database environment, you use the REDISTRIBUTE DATABASE PARTITION GROUP command.

Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

In a partitioned database you might redistribute data for the following reasons:

- To balance data volumes and processing loads across database partitions.

  Performance improves if data access can be spread out over more than one database partition.
- To introduce skew in the data distribution across database partitions.

  Access and throughput performance might improve if you redistribute data in a frequently accessed table so that infrequently accessed data is on a small number of database partitions in the database partition group, and the frequently accessed data is distributed over a larger number of database partitions. This would improve access performance and throughput on the most frequently run applications.

To preserve table collocation, use the REDISTRIBUTE DATABASE PARTITION GROUP command to redistribute data at the database partition group level. All tables are redistributed in a single operation. To achieve a specified data distribution, the REDISTRIBUTE DATABASE PARTITION GROUP command divides tables among the database database partitions as it moves the rows. Depending on the option you specify, the utility can either generate a target distribution map or use an existing distribution map as input.

How data is redistributed across database partitions

Data redistribution is performed on the set of tables in the specified database partition group of a database. You must connect to the database at the catalog database partition before executing REDISTRIBUTE DATABASE PARTITION GROUP command to invoke the Data Redistribution utility. The utility uses both the source distribution map and the target distribution map to identify which hash database partitions have been assigned to a new location, which is a new database partition number. All rows that correspond to a database partition that has a new location are moved from the database partition specified in the source distribution map to the database partition specified in the target distribution map.

The Data Redistribution utility performs the following steps:

1. Obtains a new distribution map ID for the target distribution map, and inserts it into the SYSCAT.PARTITIONMAPS catalog view.

2. Updates the REBALANCE_PMAP_ID column in the SYSCAT.DBPARTITIONGROUPS catalog view for the database partition group with the new distribution map ID.
3. Adds any new database partitions to the SYSCAT.DBPARTITIONGROUPDEF catalog view.
4. Sets the IN_USE column in the SYSCAT.DBPARTITIONGROUPDEF catalog view to ’D’ for any database partition that is to be dropped.
5. Does a COMMIT for the catalog updates.
6. Creates database files for all new database partitions.
7. Redistributes the data on a table-by-table basis for every table in the database partition group, in the following steps:
   a. Locks the row for the table in the SYSTABLES catalog table.
   b. Invalidates all packages that involve this table. The distribution map ID associated with the table changes because the table rows are redistributed. Because the packages are invalidated, the compiler must obtain the new database partitioning information for the table and generate packages accordingly.
   c. Locks the table in exclusive mode.
   d. Uses DELETEs and INSERTs to redistribute the data in the table.
   e. If the redistribution operation succeeds, it issues a COMMIT for the table and continues with the next table in the database partition group. If the operation fails before the table is fully redistributed, the utility Issues a ROLLBACK on updates to the table, ends the entire redistribution operation and returns an error.
8. Deletes database files and deletes entries in the SYSCAT.NODEGROUPDEF catalog view for database partitions that were previously marked to be dropped.
9. Updates the database partition group record in the SYSCAT.NODEGROUPS catalog view to set PMAP_ID to the value of REBALANCE_PMAP_ID and REBALANCE_PMAP_ID to NULL.
10. Deletes the old distribution map from the SYSCAT.PARTITIONMAPS catalog view.
11. Does a COMMIT for all changes.

Related concepts:
- “Log space requirements for data redistribution” on page 341
- “Redistribution error recovery” on page 342

Related tasks:
- “Determining whether to redistribute data” on page 338
- “Redistributing data across database partitions” on page 339

Determining whether to redistribute data

Before you decide to redistribute data, find out whether data is distributed unequally among database partitions. After you have current distribution information, you can use this information to create a custom redistribution file or distribution map.

Procedure:

To get information about current data distributions for database partitions in a database partition group:

1. Determine if any database partitions have unequal distributions of rows.

   For the largest table, use an appropriate database partitioning column and enter a query such as the following:

   ```
   SELECT PARTITION(column_name), COUNT(*) FROM table_name
      GROUP BY PARTITION(column_name)
      ORDER BY PARTITION(column_name) DESC
      FETCH FIRST 100 ROWS ONLY
   ```

   The PARTITION and DBPARTITIONNUM SQL functions determine the current data distribution across hash partitions or database partitions. The PARTITION function returns the distribution map index for each row of the table. The DBPARTITIONNUM function returns the database partition number of the row.
2. Execute this query for other large tables that are distributed across the database partition group.
3. Use the information to create both a distribution file and a target distribution map.

Related concepts:

- "Data redistribution" on page 337
- "Log space requirements for data redistribution" on page 341

Related tasks:

- "Redistributing data across database partitions" on page 339

Redistributing data across database partitions

In a partitioned database environment, you might redistribute data among database partitions to balance data access in the following cases:

- When some database partitions contain more data than others
- When some database partitions are accessed more frequently than others

Prerequisites:

Log file size: Ensure that log files are large enough for the data redistribution operation. The log file on each affected database partition must be large enough to accommodate the INSERT and DELETE operations performed there.

Replicated materialized query tables: If the data in a database partition group contains replicated materialized query tables, you must drop these tables before you redistribute the data. After data is redistributed, you can recreate the materialized query tables.

Restrictions:

You can do the following operations on objects of the database partition group while the utility is running. You cannot, however, do them on the table that is being redistributed. You can:

- Create indexes on other tables. The CREATE INDEX statement uses the distribution map of the affected table.
- Drop other tables. The DROP TABLE statement uses the distribution map of the affected table.

- Drop indexes on other tables. The DROP INDEX statement uses the distribution map of the affected table.
- Query other tables.
- Update other tables.
- Create new tables in a table space defined in the database partition group. The CREATE TABLE statement uses the target distribution map.
- Create table spaces in the database partition group.

You cannot do the following operations while the utility is running:
- Start another redistribution operation on the database partition group
- Execute an ALTER TABLE statement on any table in the database partition group
- Drop the database partition group
- Alter the database partition group.

You cannot use this procedure to redistribute data after adding a database partition to a single-partition system unless all affected tables have a distribution key. The REDISTRIBUTE DATABASE PARTITION GROUP command relies on distribution keys to redistribute data. The distibution key is generated automatically when a table is created in a multi-partition database partition group, or can be explicitly defined using the CREATE TABLE or ALTER TABLE SQL statements. If your tables were created in a single-partition partition group, and you did not define the distribution key in the CREATE TABLE SQL statement, there will be no distribution keys defined. You must use the ALTER TABLE SQL statement to create a distribution key for each affected table before redistributing the data.

Procedure:

To redistribute data across database partitions in a database partition group:
1. Connect to the database partition that contains the system catalog tables.
2. Perform prerequisite tasks, if necessary.
3. Issue the REDISTRIBUTE DATABASE PARTITION GROUP command.

   Note: In previous versions of DB2, this command used the NODEGROUP keyword instead of the DATABASE PARTITION GROUP keywords.

   Specify the following arguments:

   database partition group name
   You must specify the database partition group within which data is to be redistributed.

   UNIFORM
   If data is evenly distributed and is to remain evenly distributed, either specify UNIFORM or omit any distribution-type argument. UNIFORM is the default.

   USING DISTFILE distfile-name
   To specify a custom distribution that corrects or creates data skew, include the distribution file name. The Redistribute Data utility uses this file to construct a target distribution map.

   USING TARGETMAP targetmap-name
   The Redistribute Data utility uses the specified target map directly.

For details, refer to the REDISTRIBUTE DATABASE PARTITION GROUP command-line utility information.

4. After redistribution is complete:
   - Recreate any replicated materialized query tables dropped before redistribution.
   - Execute the RUNSTATS command to collect data distribution statistics for the SQL compiler and optimizer to use when it chooses data access plans for queries.

Note: The Explain tables contain information about the distribution map used to redistribute data.

You can also call the sqludrdt API from a client application to redistribute data across a database partition group.

Related concepts:

- "Data redistribution" on page 337
- "Log space requirements for data redistribution" on page 341
- "Redistribution error recovery" on page 342

Related tasks:

- "Determining whether to redistribute data" on page 338

Related reference:

- "Restrictions on native XML data store" in *XML Guide*
- "sqludrdt - Redistribute data across a database partition group" on page 820

Log space requirements for data redistribution

Before you redistribute data across database partitions, consider the log-space requirements.

The log must be large enough to accommodate the INSERT and DELETE operations at each database partition where data is being redistributed. The heaviest logging requirements will be either on the database partition that will lose the most data, or on the database partition that will gain the most data.

If you are moving to a larger number of database partitions, use the ratio of current database partitions to the new number of database partitions to estimate the number of INSERT and DELETE operations. For example, consider redistributing data that is uniformly distributed before redistribution. If you are moving from four to five database partitions, approximately twenty percent of the four original database partitions will move to the new database partition. This means that twenty percent of the DELETE operations will occur on each of the four original database partitions, and all of the INSERT operations will occur on the new database partition.

Consider a non-uniform distribution of the data, such as the case in which the distribution key contains many NULL values. In this case, all rows that contain a NULL value in the distribution key move from one database partition under the old distribution scheme and to a different database partition under the new distribution scheme. As a result, the amount of log space required on those two database partitions increases, perhaps well beyond the amount calculated by assuming uniform distribution.

The redistribution of each table is a single transaction. For this reason, when you estimate log space, you multiply the percentage of change, such as twenty percent, by the size of the largest table. Consider, however, that the largest table might be uniformly distributed but the second largest table, for example, might have one or more inflated database partitions. In such a case, consider using the non-uniformly distributed table instead of the largest one.

Note: After you estimate the maximum amount of data to be inserted and deleted at a database partition, double that estimate to determine the peak size of the active log. If this estimate is greater than the active log limit of 256 GB, then the data redistribution must be done in steps. Use the “makepmap” utility to generate a series of target distribution maps, one for each step. You might also set the *logsecond* database configuration parameter to `-1` to avoid most log space problems.

Related concepts:

- “Data redistribution” on page 337

Redistribution error recovery

After the redistribution operation begins to execute, a file is written to the `redist` subdirectory of the `sqllib` directory. This status file lists any operations that are done on database partitions, the names of the tables that were redistributed, and the completion status of the operation. If a table cannot be redistributed, its name and the applicable SQLCODE is listed in the file. If the redistribution operation cannot begin because of an incorrect input parameter, the file is not written and an SQLCODE is returned.

The file has the following naming convention:

```
For UNIX platforms:
  databasename.database partition groupname.timestamp
For non-UNIX platforms:
  databasename\database partition groupname\date\time
```

Note: On non-UNIX platforms, only the first eight (8) bytes of the database partition group name are used.

If the data redistribution operation fails, some tables may be redistributed, while others are not. This occurs because data redistribution is performed a table at a time. You have two options for recovery:

- Use the CONTINUE option to continue the operation to redistribute the remaining tables.
- Use the ROLLBACK option to undo the redistribution and set the redistributed tables back to their original state. The rollback operation can take about the same amount of time as the original redistribution operation.

Before you can use either option, a previous data redistribution operation must have failed such that the REBALANCE_PMID column in the SYSCAT.DBPARTITIONGROUPS table is set to a non-NULL value.

If you happen to delete the status file by mistake, you can still attempt a CONTINUE operation.

Related concepts:

- "Data redistribution" on page 337

Related tasks:

- "Redistributing data across database partitions" on page 339

Redistributing data using step-wise redistribute procedures

The stepwise redistribute stored procedures can be used to safely redistribute a database partition group in a number of steps:

1. Analyze the database partition group regarding log space availability and data skew using ANALYZE_LOG_SPACE procedure - Retrieve log space analysis information.

   The analyze_log_space function returns a result set (an open cursor) of the log space analysis results, containing fields for each of the database partitions of the given database partition group.
2. Create data distribution file for a given table using GENERATE_DISTFILE procedure - Generate a data distribution file

   The generate_Distfile function generates a data distribution file for the given table and saves it under the given fileName.
3. Create and report the content of a stepwise redistribution plan for the database partition group using STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group
4. Create data distribution file for a given table using GET_SWRD_SETTINGS procedure - Retrieve redistribute information and SET_SWRD_SETTINGS procedure - Create or change redistribute registry.

   The get_swrd_settings function reads the existing redistribute registry records for the given database partition group.

   The set_swrd_settings function creates or make changes to the redistribute registry. If the registry does not exist, it creates it and add records into it. If the registry already exists, it uses overwriteSpec to identify which of the field values need to be overwritten. The overwriteSpec field enables this function to take NULL inputs for the fields that do not need to be updated.
5. Redistribute the database partition group according to the plan using STEPWISE_REDISTRIBUTE_DBPG procedure - Redistribute part of database partition group.

   The stepwise_redistribute_dbpg function redistributes part of the database partition group according to the input and the setting file.

Usage example

The following is an example of a CLP script on AIX:

```
# -------------------------------------------------------------------------------------
# Set the database you wish to connect to
# -------------------------------------------------------------------------------------
dbName="SAMPLE"

# -------------------------------------------------------------------------------------
# Set the target database partition group name
# -------------------------------------------------------------------------------------
dbpgName="IBMDEFAULTGROUP"
```

```
# ------------------------------------------------------------------------------------------
# Specify the table name and schema
# ------------------------------------------------------------------------------------------
tbSchema="$USER"
tbName="STAFF"

# ------------------------------------------------------------------------------------------
# Specify the name of the data distribution file
# ------------------------------------------------------------------------------------------
distFile="$HOME/sqllib/function/$dbName.IBMDEFAULTGROUP_swrdData.dst"

export DB2INSTANCE=$USER
export DB2COMM=TCPIP

# ------------------------------------------------------------------------------------------
# Invoke call statements in clp
# ------------------------------------------------------------------------------------------
db2start
db2 -v "connect to $dbName"

# ------------------------------------------------------------------------------------------
# Analysing the effect of adding a database partition without applying the changes - a 'what if'
# hypothetical analysis
#
# - In the following case, the hypothesis is adding database partition 40, 50 and 60 to the
#   database partition group, and for database partitions 10,20,30,40,50,60, using a respective
#   target ratio of 1:2:1:2:1:2.
#
# NOTE: in this example only partitions 10, 20 and 30 actually exist in the database
#       partition group
# ------------------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'A', '40,50,60', '10,20,30,40,50,60', '1,2,1,2,1,2')"

# ------------------------------------------------------------------------------------------
# Analysing the effect of dropping a database partition without applying the changes
#
# - In the following case, the hypothesis is dropping database partition 30 from the database
#   partition group, and redistributing the data in database partitions 10 and 20 using a
#   respective target ratio of 1 : 1
#
# NOTE: In this example all database partitions 10, 20 and 30 should exist in the database
#       partition group
# ------------------------------------------------------------------------------------------
db2 -v "call sysproc.analyze_log_space('$dbpgName', '$tbSchema', '$tbName', 2, ' ',
'D', '30', '10,20','1,1')"

# ------------------------------------------------------------------------------------------
# Generate a data distribution file to be used by the redistribute process
# ------------------------------------------------------------------------------------------
db2 -v "call sysproc.generate_distfile('$tbSchema', '$tbName', '$distFile')"

# ------------------------------------------------------------------------------------------
# Write a step wise redistribution plan into a registry
#
# Setting the 10th parameter to 1, may cause a currently running step wise redistribute
# stored procedure to complete the current step and stop, until this parameter is reset
# to 0, and the redistribute stored procedure is called again.
# ------------------------------------------------------------------------------------------
db2 -v "call sysproc.set_swrd_settings('$dbpgName', 255, 0, ' ', '$distFile', 1000,
12, 2, 1, 0, '10,20,30', '50,50,50')"

# ------------------------------------------------------------------------------------------
# Report the content of the step wise redistribution plan for the given database
# partition group.
# ------------------------------------------------------------------------------------------
```

```
db2 -v "call sysproc.get_swrd_settings('$dbpgName', 255, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"

# ---------------------------------------------------------------------------------------
# Redistribute the database partition group "dbpgName" according to the redistribution
# plan stored in the registry by set_swrd_settings. It starting with step 3 and
# redistributes the data until 2 steps in the redistribution plan are completed.
# ---------------------------------------------------------------------------------------
db2 -v "call sysproc.stepwise_redistribute_dbpg('$dbpgName', 3, 2)"
```

Related concepts:

- "Data redistribution" on page 337

Chapter 29. Using Windows database partition servers

When working to change the characteristics of your configuration in a Windows environment, the tasks involved are carried out using specific utilities.

The utilities presented here are:
- "Listing database partition servers in an instance"
- "Adding a database partition server to an instance (Windows)"
- "Changing the database partition (Windows)" on page 349
- "Dropping a database partition from an instance (Windows)" on page 350

Listing database partition servers in an instance

Procedure:

On Windows, use the `db2nlist` command to obtain a list of database partition servers that participate in an instance.

The command is used as follows:

```
db2nlist
```

When using this command as shown, the default instance is the current instance (set by the DB2INSTANCE environment variable). To specify a particular instance, you can specify the instance using:

```
db2nlist /i:instName
```

where `instName` is the particular instance name you want.

You can also optionally request the status of each database partition server by using:

```
db2nlist /s
```

The status of each database partition server might be one of: starting, running, stopping, or stopped.

Related tasks:
- "Adding a database partition server to an instance (Windows)" on page 347
- "Changing the database partition (Windows)" on page 349
- "Dropping a database partition from an instance (Windows)" on page 350

Adding a database partition server to an instance (Windows)

Procedure:

On Windows, use the `db2ncrt` command to add a database partition server to an instance.

Note: Do not use the `db2ncrt` command if the instance already contains databases. Instead, use the `db2start addnode` command. This ensures that the database is correctly added to the new database partition server. **DO NOT EDIT** the `db2nodes.cfg` file, since changing the file might cause inconsistencies in the partitioned database environment.

The command has the following required parameters:

```
db2ncrt /n:node_number
        /u:username,password
        /p:logical_port
```

- /n:

  The unique database partition number to identify the database partition server. The number can be from 1 to 999 in ascending sequence.
- /u:

  The logon account name and password of the DB2 service.
- /p:logical_port

  The logical port number used for the database partition server if the logical port is not zero (0). If not specified, the logical port number assigned is 0.

The logical port parameter is only optional when you create the first database partition on a computer. If you create a logical database partition, you must specify this parameter and select a logical port number that is not in use. There are several restrictions:

- On every computer there must be a database partition server with a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the services file in `%SystemRoot%\system32\drivers\etc` directory. For example, if you reserve a range of four ports for the current instance, then the maximum port number would be 3 (ports 1, 2, and 3; port 0 is for the default logical database partition). The port range is defined when `db2icrt` is used with the `/r:base_port, end_port` parameter.

There are also several optional parameters:

- /g:network_name

  Specifies the network name for the database partition server. If you do not specify this parameter, DB2 uses the first IP address it detects on your system.

  Use this parameter if you have multiple IP addresses on a computer and you want to specify a specific IP address for the database partition server. You can enter the *network_name* parameter using the network name or IP address.
- /h:host_name

  The TCP/IP host name that is used by FCM for internal communications if the host name is not the local host name. This parameter is required if you add the database partition server on a remote computer.
- /i:instance_name

  The instance name; the default is the current instance.
- /m:computer_name

  The computer name of the Windows workstation on which the database partition resides; the default name is the computer name of the local computer.
- /o:instance_owning_computer

  The computer name of the computer that is the instance-owning computer; the default is the local computer. This parameter is required when the `db2ncrt` command is invoked on any computer that is not the instance-owning computer.

For example, if you want to add a new database partition server to the instance TESTMPP (so that you are running multiple logical database partitions) on the instance-owning computer MYMACHIN, and you want this new database partition to be known as database partition 2 using logical port 1, enter:

```
db2ncrt /n:2 /p:1 /u:my_id,my_pword /i:TESTMPP
   /M:TEST /o:MYMACHIN
```

Related reference:
- "db2icrt - Create instance " on page 479
- "db2ncrt - Add database partition server to an instance " on page 486
- "db2start - Start DB2 command" in *Command Reference*

Changing the database partition (Windows)

Procedure:

On Windows, use the db2nchg command to do the following:
- Move the database partition from one computer to another.
- Change the TCP/IP host name of the computer.

  If you are planning to use multiple network adapters, you must use this command to specify the TCP/IP address for the "netname" field in the *db2nodes.cfg* file.
- Use a different logical port number.
- Use a different name for the database partition server.

The command has the following required parameter:

```
db2nchg /n:node_number
```

The parameter /n: is the number of the database partition server's configuration you want to change. This parameter is required.

Optional parameters include:
- /i:instance_name

  Specifies the instance that this database partition server participates in. If you do not specify this parameter, the default is the current instance.
- /u:username,password

  Changes the logon account name and password for the DB2 database service. If you do not specify this parameter, the logon account and password remain the same.
- /p:logical_port

  Changes the logical port for the database partition server. This parameter must be specified if you move the database partition server to a different computer. If you do not specify this parameter, the logical port number remains unchanged.
- /h:host_name

  Changes the TCP/IP hostname used by FCM for internal communications. If you do not specify this parameter, the hostname is unchanged.
- /m:computer_name

  Moves the database partition server to another computer. The database partition server can only be moved if there are no existing databases in the instance.
- /g:network_name

Changes the network name for the database partition server.

Use this parameter if you have multiple IP addresses on a computer and you want to use a specific IP address for the database partition server. You can enter the `network_name` using the network name or the IP address.

For example, to change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to use the logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

The DB2 database manager provides the capability of accessing DB2 database system registry variables at the instance level on a remote computer. Currently, DB2 database system registry variables are stored in three different levels: computer or global level, instance level, and database partition level. The registry variables stored at the instance level (including the database partition level) can be redirected to another computer by using DB2REMOTEPREG. When DB2REMOTEPREG is set, the DB2 database manager will access the DB2 database system registry variables from the computer pointed to by DB2REMOTEPREG. The db2set command would appear as:

```
db2set DB2REMOTEPREG=<remote workstation>
```

where <remote workstation> is the remote workstation name.

Note:

- Care should be taken in setting this option since all DB2 database instance profiles and instance listings will be located on the specified remote computer name.
- If your environment includes users from domains, ensure that the logon account associated with the DB2 instance service is a domain account. This ensures that the DB2 instance has the appropriate privileges to enumerate groups at the domain level.

This feature might be used in combination with setting DBINSTPROF to point to a remote LAN drive on the same computer that contains the registry.

Related concepts:

- "DB2 registry and environment variables" in *Performance Guide*

Related reference:

- "db2nchg - Change database partition server configuration " on page 485

Dropping a database partition from an instance (Windows)

Procedure:

On Windows, use the `db2ndrop` command to drop a database partition server from an instance that has no databases. If you drop a database partition server, its database partition number can be reused for a new database partition server.

Exercise caution when you drop database partition servers from an instance. If you drop the instance-owning database partition server zero (0) from the instance, the instance will become unusable. If you want to drop the instance, use the `db2idrop` command.

Note: Do not use the `db2ndrop` command if the instance contains databases. Instead, use the `db2stop drop nodenum` command. This ensures that the database is correctly removed from the database partition. **DO NOT EDIT** the `db2nodes.cfg` file, since changing the file might cause inconsistencies in the partitioned database environment.

If you want to drop a database partition that is assigned the logical port 0 from a computer that is running multiple logical database partitions, you must drop all the other database partitions assigned to the other logical ports before you can drop the database partition assigned to logical port 0. Each database partition server must have a database partition assigned to logical port 0.

The command has the following parameters:

```
db2ndrop /n:node_number /i:instance_name
```

- /n:

  The unique database partition number to identify the database partition server. This is a required parameter. The number can be from zero (0) to 999 in ascending sequence. Recall that database partition zero (0) represents the instance-owning computer.
- /i:instance_name

  The instance name. This is an optional parameter. If not given, the default is the current instance (set by the DB2INSTANCE registry variable).

Related concepts:
- "DB2 registry and environment variables" in *Performance Guide*

Related reference:
- "db2idrop - Remove instance command" in *Command Reference*
- "db2ndrop - Drop database partition server from an instance " on page 487
- "db2stop - Stop DB2 command" in *Command Reference*

Chapter 30. Multiple logical partitions

When several database partition servers are running on the same computer, the computer is said to be running multiple logical partitions. This section describes when to use and how to configure multiple logical partitions.

When to use multiple logical partitions

Typically, you configure DB2 Enterprise Server Edition to have one database partition server assigned to each computer. There are several situations, however, in which it would be advantageous to have several database partition servers running on the same computer. This means that the configuration can contain more database partitions than computers. In these cases, the computer is said to be running *multiple logical partitions* or *multiple logical nodes* if they participate in the *same* instance. If they participate in different instances, this computer is *not* hosting multiple logical partitions.

With multiple logical partition support, you can choose from three types of configurations:

- A standard configuration, where each computer has only one database partition server.
- A multiple logical partition configuration, where a computer has more than one database partition server.
- A configuration where several logical partitions run on each of several computers.

Configurations that use multiple logical partitions are useful when the system runs queries on a computer that has symmetric multiprocessor (SMP) architecture. The ability to configure multiple logical partitions on a computer is also useful if a computer fails. If a computer fails (causing the database partition server or servers on it to fail), you can restart the database partition server (or servers) on another computer using the DB2START NODENUM command. This ensures that user data remains available.

Another benefit is that multiple logical partitions can exploit SMP hardware configurations. In addition, because database partitions are smaller, you can obtain better performance when performing such tasks as backing up and restoring database partitions and table spaces, and creating indexes.

Related tasks:

- “Configuring multiple logical partitions” on page 353

Related reference:

- “db2start - Start DB2 command” in *Command Reference*

Configuring multiple logical partitions

Procedure:

You can configure multiple logical partitions in one of two ways:

- Configure the logical partitions (database partitions) in the `db2nodes.cfg` file. You can then start all the logical and remote partitions with the DB2START command or its associated API.

  Note: For Windows, you must use *db2ncrt* to add a database partition if there is no database in the system; or, DB2START ADDNODE command if there is one or more databases. Within Windows, the *db2nodes.cfg* file should never be manually edited.
- Restart a logical partition on another processor on which other logical partitions (nodes) are already running. This allows you to override the hostname and port number specified for the logical partition in `db2nodes.cfg`.

To configure a logical partition (node) in `db2nodes.cfg`, you must make an entry in the file to allocate a logical port number for the database partition. Following is the syntax you should use:

```
nodenumber hostname logical-port netname
```

Note: For Windows, you must use *db2ncrt* to add a database partition if there is no database in the system; or, DB2START ADDNODE command if there is one or more databases. Within Windows, the *db2nodes.cfg* file should never be manually edited.

The format for the *db2nodes.cfg* file on Windows is different when compared to the same file on Unix. On Windows, the column format is:

```
nodenumber hostname computername logical_port netname
```

Use the fully-qualified name for the hostname. The /etc/hosts file also should use the fully-qualified name. If the fully-qualified name is not used in the db2nodes.cfg file and in the /etc/hosts file, you might receive error message SQL30082N RC=3.

You must ensure that you define enough ports in the `services` file of the `etc` directory for FCM communications.

Related concepts:
- "When to use multiple logical partitions" on page 353

Related tasks:
- "Changing node and database configuration files" in *Administration Guide: Implementation*
- "Creating a node configuration file" on page 287

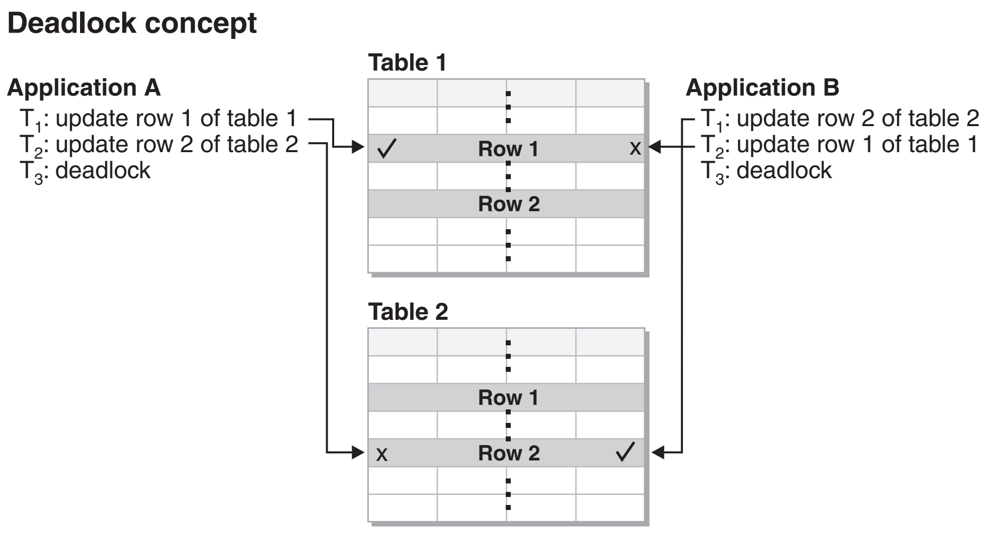
Related reference:
- "db2ncrt - Add database partition server to an instance " on page 486
- "db2start - Start DB2 command" in *Command Reference*

Part 8. Concurrency, isolation levels, and locking

Chapter 31. Concurrency, Isolation Levels, and Locking

Deadlocks

A deadlock is created when two applications are each locking data needed by the other, resulting in a situation when neither application can continue execution. For example, in the following diagram, there are two applications are running concurrently: Application A and Application B. The first step of application A is to update the first row of Table 1, and the second step is to update the second row of Table 2. Application B updates the second row of Table 2 first, and then the first row of Table 1. At one point in time, T1, Application A is executing its first step, locking the first row of Table 1 to update it. At the same time, Application B locks the second row in Table 2 to make an update. At T2, Application A tries to execute the next step and requests a lock on the second row in Table 2 for an update. However, at the same time, Application B is trying to lock and update the first row in Table 1. Since Application A will not release its lock on the first row of Table 1 until it is able to complete an update of the second row in Table 2, and Application B will not release its lock on the second row on Table 2 until it can lock and update the first row of Table 1, a deadlock occurs. The applications can wait forever until one application releases the lock on the held data.



Deadlock concept
Table 1
Application A
T1: update row 1 of table 1
T2: update row 2 of table 2
T3: deadlock
Row 1
Row 2
Application B
T1: update row 2 of table 2
T2: update row 1 of table 1
T3: deadlock
Table 2
Row 1
Row 2

Figure 25. Deadlock between applications

Because applications do not voluntarily release locks on data that they need, a deadlock detector process is required to break deadlocks and allow application processing to continue. As its name suggests, the deadlock detector monitors the information about agents waiting on locks, awakening at intervals specified by the *dlchktime* configuration parameter.

If it finds a deadlock, the deadlock detector arbitrarily selects one deadlocked process as the *victim process* to roll back. The victim process is awakened, and returns SQLCODE -911 (SQLSTATE 40001), with reason code 2, to the calling application. The database manager rolls back the selected process automatically. When the rollback is complete, the locks that belonged to the victim process are released, and the other processes involved in the deadlock can continue.

To ensure good performance, select the proper interval for the deadlock detector. An interval that is too short causes unnecessary overhead, and an interval that is too long allows a deadlock to delay a process for an unacceptable amount of time. For example, a wake-up interval of 5 minutes may allow a deadlock to exist for almost 5 minutes, which can seem like a long time for short transaction processing. It is important to balance the possible delays in resolving deadlocks with the overhead of detecting them.

Notes:

1. In a partitioned database environment, the *dlchktime* configuration parameter interval is applied only at the catalog node. If a large number of deadlocks are detected in a partitioned database environment, increase the value of the *dlchktime* parameter to account for lock waits and communication waits.
2. In a partitioned database, each database partition sends *lock graphs* to the database partition that contains the system catalog views. Global deadlock detection takes place on this database partition.

A different problem occurs when an application with more than one independent process that accesses the database is structured to make deadlocks likely. For example, an application in which several processes access the same table for reads and then writes. If the processes do read-only SQL or XQuery queries and then do SQL updates on the same table, the chance of deadlocks increases because of potential contention between the processes for the same data. For instance, if two processes read the table and then update the table, process A might try to get an X lock on a row on which process B has an S lock. To avoid such deadlocks, applications that access data with the intention of modifying it should do one of the following:

- Use the FOR UPDATE OF clause when performing a select operation. This clause ensures that a U lock is imposed when process A attempts to read the data. Row blocking is disabled.
- Use the WITH RR USE AND KEEP UPDATE LOCKS clause or the WITH RS USE AND KEEP UPDATE LOCKS clause when performing the query. Either clause ensures that a U lock is imposed when process A attempts to read the data and allows row blocking.

At the same time a database is created, a detailed deadlocks event monitor is created. As with any monitor, there is some overhead associated with this event monitor.

To limit the amount of disk space that this event monitor consumes, the event monitor deactivates and a message is written to the administration notification log when it has reached its maximum number of output files. Removing output files that are no longer needed allows the event monitor to reactivate on the next database activation.

If you do not want the detailed deadlocks event monitor, the event monitor can be dropped using the command:

```
DROP EVENT MONITOR db2detaildeadlock
```

In a federated system environment in which an application accesses nicknames, the data requested by the application might not be available because of a deadlock at a data source. When this happens, DB2 relies on the deadlock handling facilities at the data source. If deadlocks occur across more than one data source, DB2 relies on data source timeout mechanisms to break the deadlock.

To log more information about deadlocks, set the database manager configuration parameter *diaglevel* to four. The logged information includes the locked object, the lock mode, and the application holding the lock. The current dynamic SQL and XQuery statements or static package names might also be logged. Dynamic SQL and XQuery statements are logged only at *diaglevel* four.

Related concepts:

- Chapter 1, "DB2 architecture and process overview," on page 3
- "Lock granularity" on page 370

Concurrency Control and Isolation Levels

Concurrency issues

Because many users access and change data in a relational database, the database manager must be able both to allow users to make these changes and to ensure that data integrity is preserved. *Concurrency* refers to the sharing of resources by multiple interactive users or application programs at the same time. The database manager controls this access to prevent undesirable effects, such as:

- **Lost updates**. Two applications, A and B, might both read the same row from the database and both calculate new values for one of its columns based on the data these applications read. If A updates the row with its new value and B then also updates the row, the update performed by A is lost.

- **Access to uncommitted data**. Application A might update a value in the database, and application B might read that value before it was committed. Then, if the value of A is not later committed, but backed out, the calculations performed by B are based on uncommitted (and presumably invalid) data.
- **Nonrepeatable reads**. Some applications involve the following sequence of events: application A reads a row from the database, then goes on to process other requests. In the meantime, application B either modifies or deletes the row and commits the change. Later, if application A attempts to read the original row again, it receives the modified row or discovers that the original row has been deleted.
- **Phantom Read Phenomenon**. The phantom read phenomenon occurs when:
  1. Your application executes a query that reads a set of rows based on some search criterion.
  2. Another application inserts new data or updates existing data that would satisfy your application's query.
  3. Your application repeats the query from step 1 (within the same unit of work).

     Some additional ("phantom") rows are returned as part of the result set that were not returned when the query was initially executed (step 1).

Note: Declared temporary tables have no concurrency issues because they are available only to the application that declared them. This type of table only exists from the time that the application declares it until the application completes or disconnects.

Concurrency control in federated database systems

A *federated database system* supports applications and users submitting SQL statements that reference two or more database management systems (DBMSs) or databases in a single statement. To reference the data sources, which consist of a DBMS and data, DB2 uses *nicknames*. Nicknames are aliases for objects in other database managers. In a federated system, DB2 relies on the concurrency control protocols of the database manager that hosts the requested data.

A DB2 federated system provides *location transparency* for database objects. For example, with location transparency if information about tables and views is moved, references to that information through nicknames can be updated without changing applications that request the information. When an application accesses data through nicknames, DB2 relies on the concurrency control protocols of data-source database managers to ensure isolation levels. Although DB2 tries to match the requested level of isolation at the data source with a logical equivalent, results may vary depending on data source capabilities.

Related concepts:
- "Isolation levels and performance" on page 361

Related tasks:
- "Specifying the isolation level" on page 364

Related reference:
- "locklist - Maximum storage for lock list " on page 1508
- "maxlocks - Maximum percent of lock list before escalation " on page 1512

Isolation levels and performance

An *isolation level* determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. Applications that use a cursor declared with a DECLARE CURSOR statement using the WITH HOLD clause will keep the chosen isolation level for the duration of the unit of work in which the OPEN CURSOR was performed. DB2 supports the following isolation levels:

- Repeatable Read
- Read Stability
- Cursor Stability
- Uncommitted Read.

Note: Some host database servers support the *no commit* isolation level. On other databases, this isolation level behaves like the uncommitted read isolation level.

Detailed explanations for each of the isolation levels follows in decreasing order of performance impact, but in increasing order of care required when accessing and updating data.

Repeatable Read

Repeatable Read (RR) locks all the rows an application references within a unit of work. Using Repeatable Read, a SELECT statement issued by an application twice within the same unit of work in which the cursor was opened returns the same result each time. With Repeatable Read, lost updates, access to uncommitted data, and phantom rows are not possible.

The Repeatable Read application can retrieve and operate on the rows as many times as needed until the unit of work completes. However, no other applications can update, delete, or insert a row that would affect the result table, until the unit of work completes. Repeatable Read applications cannot see uncommitted changes of other applications.

With Repeatable Read, every row that is referenced is locked, not just the rows that are retrieved. Appropriate locking is performed so that another application cannot insert or update a row that would be added to the list of rows referenced by a query if that query were to be re-executed. This prevents phantom rows from occurring. For example, if you scan 10 000 rows and apply predicates to them, locks are held on all 10 000 rows, even though only 10 rows qualify.

Note: The Repeatable Read isolation level ensures that all returned data remains unchanged until the time the application sees the data, even when temporary tables or row blocking are used.

Since Repeatable Read may acquire and hold a considerable number of locks, these locks may exceed the number of locks available as a result of the *locklist* and *maxlocks* configuration parameters. In order to avoid lock escalation, the optimizer may elect to acquire a single table-level lock immediately for an index scan, if it believes that lock escalation is very likely to occur. This functions as though the database manager has issued a LOCK TABLE statement on your behalf. If you do not want a table-level lock to be obtained ensure that enough locks are available to the transaction or use the Read Stability isolation level.

When evaluating referential constraints, there are a few situations where DB2 will internally upgrade the isolation level used on the scan on the child table to Repeatable Read (RR), regardless of the isolation level set by the user. This will result in additional locks being held until commit, which increases the likelihood of a deadlock or lock timeout occurring. To avoid this, it is recommended that you create an index that only contains the column or columns of the foreign key, allowing the RI scan to use this index instead.

Read Stability

Read Stability (RS) locks only those rows that an application retrieves within a unit of work. It ensures that any qualifying row read during a unit of work is not changed by other application processes until the unit of work completes, and that any row changed by another application process is not read until the change is committed by that process. That is, "nonrepeatable read" behavior is **not** possible.

Unlike repeatable read, with Read Stability, if your application issues the same query more than once, you may see additional *phantom* rows (the *phantom read phenomenon*). Recalling the example of scanning 10 000 rows, Read Stability only locks the rows that qualify. Thus, with Read Stability, only 10 rows are retrieved, and a lock is held only on those ten rows. Contrast this with Repeatable Read, where in this example, locks would be held on all 10 000 rows. The locks that are held can be share, next share, update, or exclusive locks.

Note: The Read Stability isolation level ensures that all returned data remains unchanged until the time the application *sees* the data, even when temporary tables or row blocking are used.

One of the objectives of the Read Stability isolation level is to provide both a high degree of concurrency as well as a stable view of the data. To assist in achieving this objective, the optimizer ensures that table level locks are not obtained until lock escalation occurs.

The Read Stability isolation level is best for applications that include all of the following:
- Operate in a concurrent environment
- Require qualifying rows to remain stable for the duration of the unit of work
- Do not issue the same query more than once within the unit of work, or do not require that the query get the same answer when issued more than once in the same unit of work.

Cursor Stability

Cursor Stability (CS) locks any row accessed by a transaction of an application while the cursor is positioned on the row. This lock remains in effect until the next row is fetched or the transaction is terminated. However, if any data on a row is changed, the lock must be held until the change is committed to the database.

No other applications can update or delete a row that a Cursor Stability application has retrieved while any updatable cursor is positioned on the row. Cursor Stability applications cannot see uncommitted changes of other applications.

Recalling the example of scanning 10 000 rows, if you use Cursor Stability, you will only have a lock on the row under your current cursor position. The lock is removed when the cursor moves off that row (unless you update that row).

With Cursor Stability, both nonrepeatable read and the phantom read phenomenon are possible. Cursor Stability is the default isolation level and should be used when you want the maximum concurrency while seeing only committed rows from other applications.

Uncommitted Read

Uncommitted Read (UR) allows an application to access uncommitted changes of other transactions. The application also does not lock other applications out of the row it is reading, unless the other application attempts to drop or alter the table. Uncommitted Read works differently for read-only and updatable cursors.

Read-only cursors can access most uncommitted changes of other transactions. However, tables, views, and indexes that are being created or dropped by other transactions are not available while the transaction is processing. Any other changes by other transactions can be read before they are committed or rolled back.

Note: Cursors that are updatable operating under the Uncommitted Read isolation level will behave as if the isolation level was cursor stability.

When it runs a program using isolation level UR, an application can use isolation level CS. This happens because the cursors used in the application program are ambiguous. The ambiguous cursors can be escalated to isolation level CS because of a BLOCKING option. The default for the BLOCKING option is UNAMBIG. This means that ambiguous cursors are treated as updatable and the escalation of the isolation level to CS occurs. To prevent this escalation, you have the following two choices:
- Modify the cursors in the application program so that they are unambiguous. Change the SELECT statements to include the FOR READ ONLY clause.
- Leave cursors ambiguous in the application program, but precompile the program or bind it with the BLOCKING ALL option to allow any ambiguous cursors to be treated as read-only when the program is run.

As in the example given for Repeatable Read, of scanning 10 000 rows, if you use Uncommitted Read, you do not acquire any row locks.

With Uncommitted Read, both nonrepeatable read behavior and the phantom read phenomenon are possible. The Uncommitted Read isolation level is most commonly used for queries on read-only tables, or if you are executing only select statements and you do not care whether you see uncommitted data from other applications.

Summary of isolation levels

The following table summarizes the different isolation levels in terms of their undesirable effects.

Table 70. Summary of isolation levels

Isolation Level	Access to uncommitted data	Nonrepeatable reads	Phantom read phenomenon
Repeatable Read (RR)	Not possible	Not possible	Not possible
Read Stability (RS)	Not possible	Not possible	Possible

Table 70. Summary of isolation levels (continued)

Isolation Level	Access to uncommitted data	Nonrepeatable reads	Phantom read phenomenon
Cursor Stability (CS)	Not possible	Possible	Possible
Uncommitted Read (UR)	Possible	Possible	Possible

The table below provides a simple heuristic to help you choose an initial isolation level for your applications. Consider this table a starting point, and refer to the previous discussions of the various levels for factors that might make another isolation level more appropriate.

Table 71. Guidelines for choosing an isolation level

Application Type	High data stability required	High data stability not required
Read-write transactions	RS	CS
Read-only transactions	RR or RS	UR

Choosing the appropriate isolation level for an application is very important to avoid the phenomena that are intolerable for that application. The isolation level affects not only the degree of isolation among applications but also the performance characteristics of an individual application since the CPU and memory resources that are required to obtain and free locks vary with the isolation level. The potential for deadlock situations also varies with the isolation level.

Related concepts:

- "Concurrency issues" on page 359

Related tasks:

- "Specifying the isolation level" on page 364

Specifying the isolation level

Because the isolation level determines how data is locked and isolated from other processes while the data is being accessed, you should select an isolation level that balances the requirements of concurrency and data integrity. The isolation level that you specify is in effect for the duration of the unit of work.

Note: Isolation levels cannot be specified for XQuery statements at the statement level.

The isolation level can be specified in several different ways. The following heuristics are used in determining which isolation level will be used in compiling an SQL or XQuery statement:

Static SQL:

- If an isolation clause is specified in the statement, then the value of that clause is used.
- If no isolation clause is specified in the statement, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Dynamic SQL:

- If an isolation clause is specified in the statement, then the value of that clause is used.
- If no isolation clause is specified in the statement, and a SET CURRENT ISOLATION statement has been issued within the current session, then the value of the CURRENT ISOLATION special register is used.
- If no isolation clause is specified in the statement, and no SET CURRENT ISOLATION statement has been issued within the current session, then the isolation level used is the one specified for the package at the time when the package was bound to the database.

Static or dynamic XQuery statements:

- The isolation level of the environment determines the isolation level when the XQuery expression is evaluated.

Note: Many commercially written applications provide a method for choosing the isolation level. Refer to the application documentation for information.

Procedure:

To specify the isolation level:

1. **At precompile or bind time:**

   For an application written in a supported compiled language, use the ISOLATION option of the command line processor PREP or BIND commands. You can also use the PREP or BIND APIs to specify the isolation level.
   - If you create a bind file at precompile time, the isolation level is stored in the bind file. If you do not specify an isolation level at bind time, the default is the isolation level used during precompilation.
   - If you do not specify an isolation level, the default of cursor stability is used.

   Note: To determine the isolation level of a package, execute the following query:
   ```
   SELECT ISOLATION FROM SYSCAT.PACKAGES
     WHERE PKGNAME = 'XXXXXXXX'
     AND PKGSCHEMA = 'YYYYYYYY'
   ```
   where *XXXXXXXX* is the name of the package and *YYYYYYYY* is the schema name of the package. Both of these names must be in all capital letters.
2. **On database servers that support REXX:**

   When a database is created, multiple bind files that support the different isolation levels for SQL in REXX are bound to the database. Other command-line processor packages are also bound to the database when a database is created.

   REXX and the command line processor connect to a database using a default isolation level of cursor stability. Changing to a different isolation level does not change the connection state. It must be executed in the CONNECTABLE AND UNCONNECTED state or in the IMPLICITLY CONNECTABLE state.

   To verify the isolation level in use by a REXX application, check the value of the SQLISL REXX variable. The value is updated every time the CHANGE SQLISL command is executed.
3. **At the statement level:**

   Use the WITH clause. The statement-level isolation level overrides the isolation level specified for the package in which the statement appears.

You can specify an isolation level for the following SQL statements:

- SELECT
- SELECT INTO
- Searched DELETE
- INSERT
- Searched UPDATE
- DECLARE CURSOR

The following conditions apply to isolation levels specified for statements:

- The WITH clause cannot be used on subqueries
- The WITH UR option applies only to read-only operations. In other cases, the statement is automatically changed from UR to CS.

4. **From CLI or ODBC at runtime:**

   Use the CHANGE ISOLATION LEVEL command. For DB2 Call Level Interface (DB2 CLI), you can change the isolation level as part of the DB2 CLI configuration. At runtime, use the *SQLSetConnectAttr* function with the SQL_ATTR_TXN_ISOLATION attribute to set the transaction isolation level for the current connection referenced by the *ConnectionHandle*. You can also use the TXNISOLATION keyword in the db2cli.ini file .

5. **When working with JDBC or SQLJ at run time:**

   Note: JDBC and SQLJ are implemented with CLI on DB2, which means the db2cli.ini settings might affect what is written and run using JDBC and SQLJ.

   In SQLJ, you use the SQLJ profile customizer (db2sqljcustomize command) to create a package. The options that you can specify for this package include its isolation level.

6. **For dynamic SQL within the current session:**

   Use the SET CURRENT ISOLATION statement to set the isolation level for dynamic SQL issued within a session. Issuing this statement sets the CURRENT ISOLATION special register to a value that specifies the level of isolation for any dynamic SQL statements issued within the current session. Once set, the CURRENT ISOLATION special register provides the isolation level for any subsequent dynamic SQL statement compiled within the session, regardless of the package issuing the statement. This isolation level will apply until the session is ended or until a SET CURRENT ISOLATION statement is issued with the RESET option.

Related concepts:

- "Concurrency issues" on page 359
- "Isolation levels" in *SQL Reference, Volume 1*

Related reference:

- "CONNECT (Type 1) " on page 1159
- "SQLSetConnectAttr function (CLI) - Set connection attributes" in *Call Level Interface Guide and Reference, Volume 2*
- "Statement attributes (CLI) list" in *Call Level Interface Guide and Reference, Volume 2*

Concurrency Control and Locking

Locks and concurrency control

To provide concurrency control and prevent uncontrolled data access, the database manager places locks on buffer pools, tables, data partitions, table blocks, or table rows. A *lock* associates a database manager resource with an application, called the *lock owner*, to control how other applications access the same resource.

The database manager uses row-level locking or table-level locking as appropriate based on:

- The isolation level specified at precompile time or when an application is bound to the database. The isolation level can be one of the following:
  – Uncommitted Read (UR)
  – Cursor Stability (CS)
  – Read Stability (RS)
  – Repeatable Read (RR)

  The different isolation levels are used to control access to uncommitted data, prevent lost updates, allow non-repeatable reads of data, and prevent phantom reads. To minimize performance impact, use the minimum isolation level that satisfies your application needs.
- The access plan selected by the optimizer. Table scans, index scans, and other methods of data access each require different types of access to the data.
- The LOCKSIZE attribute for the table. The LOCKSIZE clause on the ALTER TABLE statement indicates the granularity of the locks used when the table is accessed. The choices are either ROW for row locks, TABLE for table locks, or BLOCKINSERT for block locks on MDC tables only. When the BLOCKINSERT clause is used on an MDC table, row-level locking is performed except on an INSERT operation where block-level locking is done instead. Use the ALTER TABLE ... LOCKSIZE BLOCKINSERT statement for MDC tables when transactions perform large inserts into disjoint cells. Use the ALTER TABLE ... LOCKSIZE TABLE statement for read-only tables. This reduces the number of locks required by database activity. For partitioned tables, table locks are first acquired and then data partition locks are acquired, as dictated by the data accessed.
- The amount of memory devoted to locking. The amount of memory devoted to locking is controlled by the `locklist` database configuration parameter. If the lock list fills, performance can degrade due to lock escalations and reduced concurrency on shared objects in the database. If lock escalations occur frequently, increase the value of either `locklist` or `maxlocks`, or both. Also, to reduce number of locks held at one time, ensure that transactions COMMIT frequently to free held locks.

Although most locking occurs on tables, when a buffer pool is created, altered, or dropped, a buffer pool lock is set. The mode used with this lock is EXCLUSIVE (X). You may encounter this type of lock when collecting system monitoring data. When viewing the snapshot, you will see that the lock name used is the identifier (ID) of the buffer pool itself.

In general, row-level locking is used unless one of the following is the case:

- The isolation level chosen is uncommitted read (UR).

- The isolation level chosen is repeatable read (RR) and the access plan requires a scan with no predicates.
- The table LOCKSIZE attribute is “TABLE”.
- The lock list fills, causing escalation.
- There is an explicit table lock acquired via the LOCK TABLE statement. The LOCK TABLE statement prevents concurrent application processes from either changing a table or using a table.

If this is an MDC table, there are several other cases where block-level locking is used instead of row-level locking, including:
- The table LOCKSIZE attribute is “BLOCKINSERT”
- The isolation level chosen is repeatable read (RR) and the access plan involves predicates
- A searched update or delete operation that involves only predicates on dimension columns

Lock escalations reduce concurrency. Conditions that might cause lock escalations should be avoided.

The duration of row locking varies with the isolation level being used:
- UR scans: No row locks are held unless row data is changing.
- CS scans: Row locks are only held while the cursor is positioned on the row.
- RS scans: Only qualifying row locks are held for the duration of the transaction.
- RR scans: All row locks are held for the duration of the transaction.

Related concepts:
- “Preventing lock-related performance issues” on page 371
- “Lock attributes” on page 368
- “Lock granularity” on page 370

Related reference:
- “Lock modes for table and RID index scans of MDC tables” on page 379
- “Lock type compatibility” on page 374
- “Lock modes and access paths for standard tables” on page 375
- “Locking for block index scans for MDC tables” on page 383
- “diaglevel - Diagnostic error capture level configuration parameter” in *Performance Guide*
- “locktimeout - Lock timeout ” on page 1510
- “locklist - Maximum storage for lock list ” on page 1508
- “dlchktime - Time interval for checking deadlock ” on page 1507

Lock attributes

Database manager locks have the following basic attributes:

Mode The type of access allowed for the lock owner as well as the type of access permitted for concurrent users of the locked object. It is sometimes referred to as the *state* of the lock.

Object
The resource being locked. The only type of object that you can lock

explicitly is a table. The database manager also imposes locks on other types of resources, such as rows, tables, and table spaces. For multidimensional clustering (MDC) tables, block locks can also be imposed; for partitioned tables, data partition locks can be imposed. The object being locked determines the *granularity* of the lock.

Duration
The length of time a lock is held. The isolation level in which the query runs affects the lock duration.

The following table shows the modes and their effects in order of increasing control over resources. For detailed information about locks at various levels, refer to the lock-mode reference tables.

Table 72. Lock Mode Summary

Lock Mode	Applicable Object Type	Description
IN (Intent None)	Table spaces, blocks, tables, data partitions	The lock owner can read any data in the object, including uncommitted data, but cannot update any of it. Other concurrent applications can read or update the table.
IS (Intent Share)	Table spaces, blocks, tables, data partitions	The lock owner can read data in the locked table, but cannot update this data. Other applications can read or update the table.
NS (Next Key Share)	Rows	The lock owner and all concurrent applications can read, but not update, the locked row. This lock is acquired on rows of a table, instead of an S lock, where the isolation level of the application is either RS or CS. NS lock mode is not used for next-key locking. It is used instead of S mode during CS and RS scans to minimize the impact of next-key locking on these scans.
S (Share)	Rows, blocks, tables, data partitions	The lock owner and all concurrent applications can read, but not update, the locked data.
IX (Intent Exclusive)	Table spaces, blocks, tables, data partitions	The lock owner and concurrent applications can read and update data. Other concurrent applications can both read and update the table.
SIX (Share with Intent Exclusive)	Tables, blocks, data partitions	The lock owner can read and update data. Other concurrent applications can read the table.
U (Update)	Rows, blocks, tables, data partitions	The lock owner can update data. Other units of work can read the data in the locked object, but cannot attempt to update it.
NW (Next Key Weak Exclusive)	Rows	When a row is inserted into an index, an NW lock is acquired on the next row. For type 2 indexes, this occurs only if the next row is currently locked by an RR scan. The lock owner can read but not update the locked row. This lock mode is similar to an X lock, except that it is also compatible with W and NS locks.
X (Exclusive)	Rows, blocks, tables, buffer pools, data partitions	The lock owner can both read and update data in the locked object. Only uncommitted read applications can access the locked object.
W (Weak Exclusive)	Rows	This lock is acquired on the row when a row is inserted into a table that does not have type-2 indexes defined. The lock owner can change the locked row. To determine if a duplicate value has been committed when a duplicate value is found, this lock is also used during insertion into a unique index. This lock is similar to an X lock except that it is compatible with the NW lock. Only uncommitted read applications can access the locked row.

Table 72. Lock Mode Summary (continued)

Lock Mode	Applicable Object Type	Description
Z (Super Exclusive)	Table spaces, tables, data partitions	This lock is acquired on a table in certain conditions, such as when the table is altered or dropped, an index on the table is created or dropped, or for some types of table reorganization. No other concurrent application can read or update the table.

Related concepts:

- "Locks and concurrency control" on page 367
- "Lock granularity" on page 370

Related reference:

- "maxlocks - Maximum percent of lock list before escalation " on page 1512
- "Lock type compatibility" on page 374
- "Lock modes and access paths for standard tables" on page 375

Lock granularity

If one application holds a lock on a database object, another application might not be able to access that object. For this reason, row-level locks, which minimize the amount of data that is locked and therefore inaccessible, are better for maximum concurrency than block-level, data partition-level or table-level locks. However, locks require storage and processing time, so a single table lock minimizes lock overhead.

The LOCKSIZE clause of the ALTER TABLE statement specifies the scope (granularity) of locks at the row, data partition, block, or table level. By default, row locks are used. Only S (Shared) and X (Exclusive) locks are requested by these defined table locks. The ALTER TABLE statement LOCKSIZE ROW clause does not prevent normal lock escalation from occurring.

A permanent table lock defined by the ALTER TABLE statement might be preferable to a single-transaction table lock using LOCK TABLE statement in the following cases:

- The table is read-only, and will always need only S locks. Other users can also obtain S locks on the table.
- The table is usually accessed by read-only applications, but is sometimes accessed by a single user for brief maintenance, and that user requires an X lock. While the maintenance program runs, the read-only applications are locked out, but in other circumstances, read-only applications can access the table concurrently with a minimum of locking overhead.

For an MDC table, you can specify BLOCKINSERT for the LOCKSIZE clause in order to use block-level locking during INSERT operations only. When this is specified, row-level locking is performed for all other operations, but only minimally for INSERT operations. That is, block-level locking is used during the insertion of rows, but row-level locking is used for next-key locking if RR scans are encountered in the indexes as they are being updated. BLOCKINSERT locking might be beneficial in the following cases:

- There are multiple transactions doing mass insertions into separate cells.

- Concurrent insertions to the same cell by multiple transactions is not occurring, or it is occurring with enough data inserted per cell by each of the transactions that the user is not concerned that each transaction will insert into separate blocks.

The ALTER TABLE statement specifies locks globally, affecting all applications and users that access that table. Individual applications might use the LOCK TABLE statement to specify table locks at an application level instead.

Related concepts:
- "Deadlocks" on page 357
- "Locks and concurrency control" on page 367

Related tasks:
- "Correcting lock escalation problems" on page 373

Related reference:
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*
- "locktimeout - Lock timeout " on page 1510

Preventing lock-related performance issues

Consider the following guidelines when you tune locking for concurrency and data integrity:

- Create small units of work with frequent COMMIT statements to promote concurrent access of data by many users.

  Include COMMIT statements when your application is logically consistent, that is, when the data you have changed is consistent. When a COMMIT is issued, locks are released except for table locks associated with cursors declared WITH HOLD.
- Specify an appropriate isolation level.

  Locks are acquired even if your application merely reads rows, so it is still important to commit read-only units of work. This is because shared locks are acquired by repeatable read, read stability, and cursor stability isolation levels in read-only applications. With repeatable read and read stability, all locks are held until a COMMIT is issued, preventing other processes from updating the locked data, unless you close your cursor using the WITH RELEASE clause. In addition, catalog locks are acquired even in uncommitted read applications using dynamic SQL or XQuery statements.

  The database manager ensures that your application does not retrieve uncommitted data (rows that have been updated by other applications but are not yet committed) unless you are using the uncommitted read isolation level.
- Use the LOCK TABLE statement appropriately.

  The statement locks an entire table. Only the table specified in the LOCK TABLE statement is locked. Parent and dependent tables of the specified table are not locked. You must determine whether locking other tables that can be accessed is necessary to achieve the desired result in terms of concurrency and performance. The lock is not released until the unit of work is committed or rolled back.

  LOCK TABLE IN SHARE MODE

  You want to access data that is *consistent in time*; that is, data current for a table at a specific point in time. If the table experiences frequent activity, the only way to ensure that the entire table remains stable is to lock it. For example, your application wants to take a snapshot of a table. However, during the time your application needs to process some rows of a table, other applications are updating rows you have not yet processed. This is allowed with repeatable read, but this action is not what you want.

As an alternative, your application can issue the LOCK TABLE IN SHARE MODE statement: no rows can be changed, regardless of whether you have retrieved them or not. You can then retrieve as many rows as you need, knowing that the rows you have retrieved have not been changed just before you retrieved them.

With LOCK TABLE IN SHARE MODE, other users can retrieve data from the table, but they cannot update, delete, or insert rows into the table.

LOCK TABLE IN EXCLUSIVE MODE

You want to update a large part of the table. It is less expensive and more efficient to prevent all other users from accessing the table than it is to lock each row as it is updated, and then unlock the row later when all changes are committed.

With LOCK TABLE IN EXCLUSIVE MODE, all other users are locked out; no other applications can access the table unless they are uncommitted read applications.

- Use ALTER TABLE statements in applications.

  The ALTER TABLE statement with the LOCKSIZE parameter is an alternative to the LOCK TABLE statement. The LOCKSIZE parameter lets you specify a lock granularity of either ROW locks or TABLE locks for the next table access. For MDC tables, it also lets you specify a lock granularity of the BLOCKINSERT clause.

  The selection of ROW locks is no different from selecting the default lock size when a table is created. The selection of TABLE locks may improve query performance by limiting the number of locks that need to be acquired. However, concurrency might be reduced because all locks are on the complete table. For MDC tables, the selection of the BLOCKINSERT clause may improve the performance of INSERT operations by locking at the block level and avoiding row locks for insertions. Row-level locking is still performed for all other operations and is performed on key insertions to protect Repeatable Read (RR) scanners. The BLOCKINSERT option is useful for large insertions into cells by individual transactions. None of the LOCKSIZE choices prevent normal lock escalation.
- Close cursors to release the locks that they hold.

  When you close a cursor with the CLOSE CURSOR statement that includes the WITH RELEASE clause, the database manager attempts to release all read locks that have been held for the cursor. Table read locks are IS, S, and U table locks. Row-read locks are S, NS, and U row locks. Block-read locks are IS, S, and U block locks.

  The WITH RELEASE clause has no effect on cursors that are operating under the CS or UR isolation levels. When specified for cursors that are operating under the RS or RR isolation levels, the WITH RELEASE clause ends some of the guarantees of those isolation levels. Specifically, a RS cursor may experience the *nonrepeatable read* phenomenon, and a RR cursor may experience either the *nonrepeatable read* or *phantom read* phenomenon.

If a cursor that is originally RR or RS is reopened after being closed using the WITH RELEASE clause, then new read locks are acquired.

In CLI applications, the DB2 CLI connection attribute SQL_ATTR_CLOSE_BEHAVIOR can be used to achieve the same results as CLOSE CURSOR WITH RELEASE.

- In a partitioned database environment, when you change the configuration parameters that affecting locking, ensure that the changes are made to all of the database partitions.

Related concepts:

- "Locks and concurrency control" on page 367
- "Lock attributes" on page 368
- "Lock granularity" on page 370

Correcting lock escalation problems

The database manager can automatically escalate locks from row or block level to table level. For partitioned tables, the database manager can automatically escalate locks from row or block level to data partition level. The *maxlocks* database configuration parameter specifies when lock escalation is triggered. The table that acquires the lock that triggers lock escalation might not be affected. Locks are first escalated for the table with the most locks, beginning with tables for which long object (LOBs) and long VARCHAR descriptors are locked, then the table with the next highest number of locks, and so on, until the number of locks held is decreased to about half of the value specified by *maxlocks*.

In a well designed database, lock escalation rarely occurs. If lock escalation reduces concurrency to an unacceptable level (indicated by the *lock_escalation* monitor element or the *db.lock_escal_rate* health indicator) you need to analyze the problem and decide how to solve it.

Prerequisites:

Ensure that lock escalation information is recorded. Set the database manager configuration parameter *notifylevel* to 3, which is the default, or to 4. At *notifylevel* of 2, only the error SQLCODE is reported. At *notifylevel* of 3 or 4, when lock escalation fails, information is recorded for the error SQLCODE and the table for which the escalation failed. The current query statement is logged only if it is a currently executing, dynamic query statement and *notifylevel*is set to 4.

Procedure:

Follow these general steps to diagnose the cause of unacceptable lock escalations and apply a remedy:

1. Analyze in the administration notification log on all tables for which locks are escalated. This log file includes the following information:
   - The number of locks currently held.
   - The number of locks needed before lock escalation is completed.
   - The table identifier information and table name of each table being escalated.
   - The number of non-table locks currently held.
   - The new table level lock to be acquired as part of the escalation. Usually, an "S," or Share lock, or an "X," or eXclusive lock is acquired.

- The internal return code of the result of the acquisition of the new table lock level.

2. Use the information in administration notification log to decide how to resolve the escalation problem. Consider the following possibilities:
   - Increase the number of locks allowed globally by increasing the value of the *maxlocks* or the *locklist* parameters, or both, in the database configuration file. In a partitioned database, make this change on all database partitions.

     You might choose this method if concurrent access to the table by other processes is most important. However, the overhead of obtaining record level locks can induce more delay to other processes than is saved by concurrent access to a table.
   - Adjust the process or processes that caused the escalation. For these processes, you might issue LOCK TABLE statements explicitly.
   - Change the degree of isolation. Note that this may lead to decreased concurrency, however.
   - Increase the frequency of commits to reduce the number of locks held at a given time.
   - Consider frequent COMMIT statements for transactions that require long VARCHAR or various kinds of long object (LOB) data. Although this kind of data is not retrieved from disk until the result set is materialized, the descriptor is locked when the data is first referenced. As a result, many more locks might be held than for rows that contain more ordinary kinds of data.

Related reference:
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*
- "maxlocks - Maximum percent of lock list before escalation " on page 1512

Lock type compatibility

Lock compatibility becomes an issue when one application currently has a lock on an object and another application requests a lock on the same object. When the two lock modes are compatible, the request for a second lock on the object can be granted.

If the lock mode of the requested lock is not compatible with the lock already held, the lock request cannot be granted. Instead, the request must wait until the first application releases its lock, and all other existing incompatible locks are released.

The following table displays information about the circumstances in which a lock request can be granted when another process holds or is requesting a lock on the same resource in a given state. A **no** indicates that the requestor must wait until all incompatible locks are released by other processes. Note that a timeout can occur when a requestor is waiting for a lock. A **yes** indicates that the lock is granted unless an earlier requestor is waiting for the resource.

Table 73. Lock Type Compatibility

	State of Held Resource											
State Being Requested	**none**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**X**	**Z**	**NW**	**W**
none	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
IN	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
IS	yes	yes	yes	yes	yes	yes	yes	yes	no	no	no	no
NS	yes	yes	yes	yes	yes	no	no	yes	no	no	yes	no
S	yes	yes	yes	yes	yes	no	no	yes	no	no	no	no
IX	yes	yes	yes	no	no	yes	no	no	no	no	no	no
SIX	yes	yes	yes	no	no	no	no	no	no	no	no	no
U	yes	yes	yes	yes	yes	no	no	no	no	no	no	no
X	yes	yes	no	no	no	no	no	no	no	no	no	no
Z	yes	no	no	no	no	no	no	no	no	no	no	no
NW	yes	yes	no	yes	no	no	no	no	no	no	no	yes
W	yes	yes	no	no	no	no	no	no	no	no	yes	no

Note:

I	Intent
N	None
NS	Next Key Share
S	Share
X	Exclusive
U	Update
Z	Super Exclusive
NW	Next Key Weak Exclusive
W	Weak Exclusive

Note:
- yes - **grant** lock requested immediately
- no - **wait** for held lock to be released or timeout to occur

Related concepts:
- "Lock attributes" on page 368
- "Locks and concurrency control" on page 367
- "Lock granularity" on page 370

Related reference:
- "Lock modes and access paths for standard tables" on page 375
- "Locking for block index scans for MDC tables" on page 383

Lock modes and access paths for standard tables

This topic includes reference information about locking methods for standard tables for different data-access plans.

The following tables list the types of locks obtained for standard tables at each level for different access plans. Each entry is made up of two parts: table lock and row lock. A dash indicates that a particular level of locking is not done.

Notes:

1. In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used.
2. Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 74. Lock Modes for Table Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 75. Lock Modes for Table Scans with Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Note: At UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks to an IS table lock and NS row locks.

Table 76. Lock Modes for RID Index Scans with no Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 77. Lock Modes for RID Index Scans with a Single Qualifying Row

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	IS/S	IX/U	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X

Table 77. Lock Modes for RID Index Scans with a Single Qualifying Row (continued)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 78. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	IS/S	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 79. Lock Modes for RID Index Scans with Index and Other Predicates (sargs, resids) Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or Delete
RR	IS/S	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

The following tables shows the lock modes for cases in which reading of the data pages is deferred to allow the list of rows to be:
- Further qualified using multiple indexes
- Sorted for efficient prefetching

Table 80. Lock modes for index scans used for deferred data page access: RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 81. Lock modes for index scans used for deferred data page access: after a RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

Table 82. Lock modes for index scans used for deferred data page access: RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 83. Lock modes for index scans used for deferred data page access: RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IS/S	IX/S		IX/X	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

Table 84. Lock modes for index scans used for deferred data page access, after a RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IS/-	IX/U	IX/X	IX/U	IX/X

Table 85. Lock modes for index scans used for deferred data page access, after a RID index scan with predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan	Update or delete
RR	IN/-	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

Related concepts:
- "Lock attributes" on page 368
- "Lock granularity" on page 370

Related reference:
- "Lock modes for table and RID index scans of MDC tables" on page 379
- "Lock type compatibility" on page 374
- "Locking for block index scans for MDC tables" on page 383

Lock modes for table and RID index scans of MDC tables

In a multi-dimensional clustering (MDC) environment, an additional lock level, BLOCK, is used. The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not used.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 86. Lock Modes for Table Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 87. Lock Modes for Table Scans with Predicates on Dimension Columns Only

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

Table 88. Lock Modes for Table Scans with Other Predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operation		Searched update or delete	
		Scan	Where current of	Scan or delete	Update
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

The following two tables show lock modes for RID indexes on MDC tables.

Table 89. Lock Modes for RID Index Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 90. Lock Modes for RID Index Scans with Single Qualifying Row

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

Table 91. Lock Modes for RID Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 92. Lock Modes for RID Index Scans with Index Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 93. Lock Modes for RID Index Scans with Other Predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Note: In the following tables, which shows lock modes for RID index scans used for deferred data-page access, at UR isolation level with IN lock for type-1 indexes or if there are predicates on include columns in the index, the isolation level is upgraded to CS and the locks are upgraded to an IS table lock, an IS block lock, and NS row locks.

Table 94. Lock modes for RID index scans used for deferred data-page access: RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 95. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

Table 96. Lock modes for RID index scans used for deferred data-page access: RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 97. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Table 98. Lock modes for RID index scans used for deferred data-page access: RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/IS/S	IX/IX/S		IX/IX/X	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

Table 99. Lock modes for RID index scans used for deferred data-page access: Deferred data-page access after a RID index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Related concepts:

- “Lock attributes” on page 368
- “Locks and concurrency control” on page 367
- “Lock granularity” on page 370

Related reference:
- “Lock modes and access paths for standard tables” on page 375
- “Locking for block index scans for MDC tables” on page 383
- “Lock type compatibility” on page 374

Locking for block index scans for MDC tables

The following tables list the types of locks obtained at each level for different access plans. Each entry is made up of three parts: table lock, block lock, and row lock. A dash indicates that a particular level of locking is not done.

Note: Lock modes can be changed explicitly with the lock-request-clause of a select statement.

Table 100. Lock Modes for Index Scans with No Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 101. Lock Modes for Index Scans with Dimension Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

Table 102. Lock Modes for Index Scans with Start and Stop Predicates Only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

Table 103. Lock Modes for Index Scans with Predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

The following table lists lock modes for block index scans used for deferred data-page access:

Table 104. Lock modes for block index scans used for deferred data-page access: Block index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 105. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with no predicates

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

Table 106. Lock modes for block index scans used for deferred data-page access: Block index scan with dimension predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

Table 107. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with dimension predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

Table 108. Lock modes for block index scans used for deferred data-page access: Block index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 109. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with start and stop predicates only

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

Table 110. Lock modes for block index scans used for deferred data-page access: Block index scan other predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

Table 111. Lock modes for block index scans used for deferred data-page access: Deferred data-page access after a block index scan with other predicates (sargs, resids)

Isolation Level	Read-only and ambiguous scans	Cursored operations		Searched update or delete	
		Scan	Where current of	Scan Delete	Update
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

Related concepts:
- "Lock granularity" on page 370

Related reference:
- "Lock modes for table and RID index scans of MDC tables" on page 379
- "Lock type compatibility" on page 374
- "Lock modes and access paths for standard tables" on page 375

Factors that affect locking

The following factors affect the mode and granularity of database manager locks:
- The type of processing that the application performs
- The data access method
- Whether indexes are type-2 or type-1
- Various configuration parameters

Related concepts:
- "Preventing lock-related performance issues" on page 371
- "Index cleanup and maintenance" in *Performance Guide*
- "Index types and next-key locking" on page 388
- "Lock attributes" on page 368
- "Locks and concurrency control" on page 367
- "Locks and data-access methods" on page 387
- "Lock granularity" on page 370
- "Locks and types of application processing" on page 386

Factors That Affect Locking

Locks and types of application processing

For the purpose of determining lock attributes, application processing can be classified as one of the following types:
- Read-only

  This type includes all select statements that are intrinsically read-only, have an explicit FOR READ ONLY clause, or are ambiguous but which the query compiler assumes to be read-only because of the value of the BLOCKING option that the PREP or BIND command specifies. This processing type requires only Share locks (S, NS, or IS).

- Intent to change

  This type includes all select statements with the FOR UPDATE clause, with the USE AND KEEP UPDATE LOCKS clause, with the USE AND KEEP EXCLUSIVE LOCKS clause, or for which the query compiler interprets an ambiguous statement to imply that change is intended. This type uses Share and Update locks (S, U, and X for rows; IX, U, X, and S for blocks; IX, U, and X for tables).
- Change

  This type includes UPDATE, INSERT, and DELETE, but not UPDATE WHERE CURRENT OF or DELETE WHERE CURRENT OF. This type requires Exclusive locks (X or IX).
- Cursor controlled

  This type includes UPDATE WHERE CURRENT OF and DELETE WHERE CURRENT OF. It also requires Exclusive locks (X or IX).

A statement that inserts, updates or deletes data in a target table, based on the result from a sub-select statement, does two types of processing. The rules for read-only processing determine the locks for the tables returned in the sub-select statement. The rules for change processing determine the locks for the target table.

Related concepts:
- "Deadlocks" on page 357
- "Preventing lock-related performance issues" on page 371
- "Index types and next-key locking" on page 388
- "Lock attributes" on page 368
- "Locks and concurrency control" on page 367
- "Locks and data-access methods" on page 387
- "Lock granularity" on page 370

Related tasks:
- "Correcting lock escalation problems" on page 373

Related reference:
- "Lock type compatibility" on page 374

Locks and data-access methods

An *access plan* is the method that the optimizer selects to retrieve data from a specific table. The access plan can have a significant effect on lock modes. For example, when an index scan is used to locate a specific row, the optimizer will probably choose row-level locking (IS) for the table. For example, if the EMPLOYEE table that has an index on employee number (EMPNO), access through an index might be used to select information for a single employee with a statement that contains the following SELECT clause:

```
SELECT *
  FROM EMPLOYEE
  WHERE EMPNO = '000310';
```

If an index is not used, the entire table must be scanned in sequence to find the selected rows, and may thus acquire a single table level lock (S). For example, if there is no index on the column SEX, a table scan might be used to select all male employees with a a statement that contains the following SELECT clause:

```
SELECT *
  FROM EMPLOYEE
  WHERE SEX = 'M';
```

Note: Cursor controlled processing uses the lock mode of the underlying cursor until the application finds a row to update or delete. For this type of processing, no matter what the lock mode of a cursor, an exclusive lock is always obtained to perform the update or delete.

Locking in range-clustered tables works slighly differently from standard key or next-key locking. In accessing a range of rows in a range-clustered table, all rows in the range are locked, even when some of those rows are empty. In standard key or next key locking, only rows with existing records are locked.

Reference tables provide detailed information about which locks are obtained for what kind of access plan.

Deferred access of the data pages implies that access to the row occurs in two steps, which results in more complex locking scenarios. The timing of lock aquisition and the persistence of the locks depend on the isolation level. Because the Repeatable Read isolation level retains all locks until the end of the transaction, the locks acquired in the first step are held and there is no need to acquire further locks in the second step. For the Read Stability and Cursor Stability isolation levels, locks must be acquired during the second step. To maximize concurrency, locks are not acquired during the first step and rely on the reapplication of all predicates to ensure that only qualifying rows are returned.

Related concepts:
- "Preventing lock-related performance issues" on page 371
- "Index types and next-key locking" on page 388
- "Lock attributes" on page 368
- "Locks and concurrency control" on page 367
- "Lock granularity" on page 370
- "Locks and types of application processing" on page 386

Related tasks:
- "Correcting lock escalation problems" on page 373

Related reference:
- "Lock modes and access paths for standard tables" on page 375
- "Lock modes for table and RID index scans of MDC tables" on page 379
- "Lock type compatibility" on page 374
- "Locking for block index scans for MDC tables" on page 383

Index types and next-key locking

As transactions cause changes to type-1 indexes, some next-key locking occurs. For type-2 indexes, minimal next-key locking occurs.

- Next-key locking for type 2 indexes

  Next-key locking occurs when a key is inserted into an index.

During insertion of a key into an index, the row that corresponds to the key that will follow the new key in the index is locked only if that row is currently locked by an RR index scan. The lock mode used for the next-key lock is NW. This next-key lock is released before the key insertion is actually performed. Key insertion occurs when a row is inserted into a table.

When updates to a row result in a change to the value of the index key for that row, key insertion also occurs because the original key value is marked deleted and the new key value is inserted into the index. For updates that affect only the include columns of an index, the key can be updated in place and no next-key locking occurs.

During RR scans, the row that corresponds to the key that follows the end of the scan range is locked in S mode. If no keys follow the end of the scan range, an end-of-table lock is acquired to lock the end of the index. If the key that follows the end of the scan range is marked deleted, the scan continues to lock the corresponding rows until it finds a key that is not marked deleted, when it locks the corresponding row for that key, or until the end of the index is locked.

- Next-key locking for type-1 indexes:

  Next-key locks occur during deletes and inserts to indexes and during index scans. When a row is updated in, deleted from, or inserted into a table, an X lock is obtained on that row. For insertions this might be downgraded to a W lock.

  When the key is deleted from the table index or inserted into it, the table row that corresponds to the key that follows the deleted or inserted key in the index is locked. For updates that affect the value of the key, the original key value is first deleted and the new value is inserted, so two next-key locks are acquired. The duration of these locks is determined as follows:

  – During index key deletion, the lock mode on the next key is X and the lock is held until commit time.
  – During index key insertion, the lock mode on the next key is NW. This lock is acquired only if there is contention for the lock, in which case the lock is released before the key is actually inserted into the index.
  – During RR scans, the table row that corresponds to the key just beyond the end of the index scan range is locked in S mode and is held until commit time.
  – During CS/RS scans, the row corresponding to the key just beyond the end of the index scan range is locked in NS mode if there is contention for the lock. This lock is released once the end of the scan range is verified.

  The next-key locking for type-1 indexes during key insertions and key deletion might result in deadlocks. The following example shows how two transactions could deadlock. With type 2 indexes, such deadlocks do not occur.

  Consider the following example of an index that contains 6 rows with the following values: `1 5 6 7 8 12`.

  1. Transaction 1 deletes the row with key value 8. The row with value 8 is locked in X mode. When the corresponding key from the index is deleted, the row with value 12 is locked in X mode.
  2. Transaction 2 deletes the row with key value 5. The row with value 5 is locked in X mode. When the corresponding key from the index is deleted, the row with value 6 is locked in X mode.
  3. Transaction 1 inserts a row with key value 4. This row is locked in W mode. When inserting the new key into the index is attempted, the row with value 6 is locked in NW mode. This lock attempt will wait on the X lock that transaction 2 has on this row.

4. Transaction 2 inserts a row with key value 9. This row is locked in W mode. When inserting the new key into the index is attempted, the row with key value 12 is locked in NW mode. This lock attempt will wait on the X lock that transaction 1 has on this row.

When type-1 indexes are used, this scenario will result in a deadlock and one of these transactions will be rolled back.

Related concepts:
- "Index cleanup and maintenance" in *Performance Guide*
- "Using relational indexes to improve performance" in *Performance Guide*
- "Index reorganization" in *Performance Guide*
- "Index structure" in *Performance Guide*
- "Online index defragmentation" in *Performance Guide*
- "Relational index performance tips" in *Performance Guide*

Evaluate uncommitted data via lock deferral

To improve concurrency, DB2 now permits the deferral of row locks for CS or RS isolation scans in some situations until a record is known to satisfy the predicates of a query. By default, when row-locking is performed during a table or index scan, DB2 locks each row that is scanned before determining whether the row qualifies for the query. To improve the concurrency of scans, it may be possible to defer row locking until after it is determined that a row qualifies for a query.

To take advantage of this feature, enable the DB2_EVALUNCOMMITTED registry variable.

With this variable enabled, predicate evaluation can occur on uncommitted data. This means that a row that contains an uncommitted update may not satisfy the query, whereas if the predicate evaluation waited until the updated transaction completed, the row may satisfy the query. Additionally, uncommitted deleted rows are skipped during table scans. DB2 will skip deleted keys in type-2 index scans if the DB2_SKIPDELETED registry variable is enabled.

These registry variable settings apply at compile time for dynamic SQL or XQuery statements and at bind time for static SQL or XQuery statements. This means that even if the registry variable is enabled at runtime, the lock avoidance strategy is not employed unless DB2_EVALUNCOMMITTED was enabled at bind time. If the registry variable is enabled at bind time but not enabled at runtime, the lock avoidance strategy is still in effect. For static SQL or XQuery statements, if a package is rebound, the registry variable setting at bind time is the setting that applies. An implicit rebind of static SQL or XQuery statements will use the current setting of the DB2_EVALUNCOMMITTED.

Applicability of evaluate uncommitted for different access plans

Table 112. RID Index Only Access

Predicates	Evaluate Uncommitted
None	No
SARGable	Yes

Table 113. Data Only Access (relational or deferred RID list)

Predicates	Evaluate Uncommitted
None	No
SARGable	Yes

Table 114. RID Index + Data Access

Predicates		Evaluate Uncommitted	
Index	**Data**	**Index access**	**Data access**
None	None	No	No
None	SARGable	No	No
SARGable	None	Yes	No
SARGable	SARGable	Yes	No

Table 115. Block Index + Data Access

Predicates		Evaluate Uncommitted	
Index	**Data**	**Index access**	**Data access**
None	None	No	No
None	SARGable	No	Yes
SARGable	None	Yes	No
SARGable	SARGable	Yes	Yes

Example

The following example provides a comparison of the default locking behavior and the new evaluate uncommitted behavior.

The table below is the ORG table from the SAMPLE database.

```
DEPTNUMB DEPTNAME       MANAGER DIVISION   LOCATION
-------- -------------- ------- ---------- -------------
      10 Head Office        160 Corporate  New York
      15 New England         50 Eastern    Boston
      20 Mid Atlantic        10 Eastern    Washington
      38 South Atlantic      30 Eastern    Atlanta
      42 Great Lakes        100 Midwest    Chicago
      51 Plains             140 Midwest    Dallas
      66 Pacific            270 Western    San Francisco
      84 Mountain           290 Western    Denver
```

The following transactions are acting on this table, with the default Cursor Stability (CS) isolation level.

Table 116. Transactions on the ORG table with the CS isolation level

SESSION 1	SESSION 2
`connect to SAMPLE`	`connect to SAMPLE`
`+c update org set  deptnumb=5 where manager=160`	
	`select * from org where deptnumb >= 10`

The uncommitted UPDATE in Session 1 holds an exclusive record lock on the first row in the table, prohibiting the SELECT query in Session 2 from returning even though the row being updated in Session 1 does not currently satisfy the query in Session 2. This is because the CS isolation level dictates that any row accessed by a query must be locked while the cursor is positioned on that row. Session 2 cannot obtain a lock on the first row until Session 1 releases its lock.

When scanning the table, the lock-wait in Session 2 can be avoided using the evaluate uncommitted feature which first evaluates the predicate and then locks the row for a true predicate evaluation. As such, the query in Session 2 would not attempt to lock the first row in the table thereby increasing application concurrency. Note that this would also mean that predicate evaluation in Session 2 would occur with respect to the uncommitted value of deptnumb=5 in Session 1. The query in Session 2 would omit the first row in its result set despite the fact that a rollback of the update in Session 1 would satisfy the query in Session 2.

If the order of operations were reversed, concurrency could still be improved with evaluate uncommitted. Under default locking behavior, Session 2 would first acquire a row lock prohibiting the searched UPDATE in Session 1 from executing even though the UPDATE in Session 1 would not change the row locked by the query of Session 2. If the searched UPDATE in Session 1 first attempted to examine rows and then only lock them if they qualified, the query in Session 1 would be non-blocking.

Restrictions

The following external restrictions apply to this new functionality:

- The registry variable DB2_EVALUNCOMMITTED must be enabled.
- The isolation level must be CS or RS.
- Row locking is to occur.
- SARGable evaluation predicates exist.
- Evaluation uncommitted is not applicable to scans on the catalog tables.
- For MDC tables, block locking can be deferred for an index scan; however, block locking will not be deferred for table scans.
- Deferred locking will not occur on a table which is executing an inplace table reorg.
- Deferred locking will not occur for an index scan where the index is type-1.
- For Iscan-Fetch plans, row locking is not deferred to the data access but rather the row is locked during index access before moving to the row in the table.
- Deleted rows are unconditionally skipped for table scans while deleted type-2 index keys are only skipped if the registry variable DB2_SKIPDELETED is enabled.

Option to disregard uncommitted insertions

The DB2_SKIPINSERTED registry variable controls whether uncommitted insertions can be ignored for cursors using the Cursor Stability (CS) or Read Stability (RS) isolation levels. The DB2 database system can handle uncommitted insertions in the following ways:

- The DB2 database system can wait until the INSERT transaction completes (commits or rolls back) and process data accordingly. This is the default option, OFF.

The following examples show instances when the default option, OFF, is preferred:

  – Suppose that two applications use a table to pass data between themselves with the first application inserting data into the table and the second one reading it. The data must be processed by the second application in the order presented in the table such that if the next row to be read is being inserted by the first application, the second application must wait until the insert is committed. In such cases, the default value for DB2_SKIPINSERTED should be used.
  – Suppose that an application modifies data by deleting the data and inserting the new image of the data. In such cases that avoid UPDATE statements, the default value for DB2_SKIPINSERTED should be used.

- The DB2 database system can ignore uncommitted insertions, which in many cases can improve concurrency. If you want this behavior, the registry variable must be specified as ON.

  In general, ON produces greater concurrency and is preferred for most applications. When the registry variable is enabled, uncommitted inserted rows are treated as if they had not yet been inserted.

Related concepts:

- "Evaluate uncommitted data via lock deferral" on page 390

Table locking during import

The import utility supports two table locking modes. The offline mode (ALLOW NO ACCESS) prevents concurrent applications from accessing table data. This is the default mode. The online mode (ALLOW WRITE ACCESS) allows concurrent applications both read and write access to the import target table.

By default, the import utility is bound to the database with isolation level RS (read stability).

Online Import (ALLOW WRITE ACCESS):

The Import utility acquires a nonexclusive (IX) lock on the target table. Holding this lock on the table has the following implications:

- If there are other applications holding an incompatible table lock, the import utility will not start inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible table lock will wait until the import commits or rolls back the current transaction. Note that import's table lock does not persist across a transaction boundary. As a result, online import has to request and potentially wait for a table lock after every commit.
- If there are other applications holding an incompatible row lock, the import utility will stop inserting data until all of these applications commit or roll back their changes.
- While import is running, any other application requesting an incompatible row lock will wait until the import operation commits or rolls back the current transaction.

To preserve the online properties, and to reduce the chance of a deadlock, online import will periodically commit the current transaction and release all row locks before escalating to an exclusive (X) table lock. Consequently, during an online import, commits might be performed even if the commitcount option was not used. A commit frequency can either be explicitly specified, or the AUTOMATIC commit mode can be used. No commits will be performed if a commitcount value of zero is explicitly specified. Note that a deadlock will occur if the concurrent application holding a conflicting row lock attempts to escalate to a table lock.

Import runs in the online mode if 'ALLOW WRITE ACCESS' is specified. The online mode is not compatible with the following:

- REPLACE, CREATE and REPLACE_CREATE import modes
- Buffered inserts
- Imports into a target view
- Imports into a hierarchy table
- Imports into a target table using table lock size

Offline Import (ALLOW NO ACCESS):

If a large number of rows is being imported into a table, the existing lock might escalate to an exclusive lock. If another application working on the same table is holding some row locks, a deadlock will occur if the lock escalates to an exclusive lock. To avoid this, the import utility requests an exclusive lock on the table at the beginning of its operation. This is the default import behavior.

Holding a lock on the table has two implications. First, if there are other applications holding a table lock, or row locks on the import target table, the import utility will wait until all of those applications commit or roll back their changes. Second, while import is running, any other application requesting locks will wait until the import operation has completed. Import runs in the offline mode if 'ALLOW WRITE ACCESS' is not specified.

Related concepts:

- "Table locking, table states and table space states" in *Data Movement Utilities Guide and Reference*

Part 9. DB2 commands

Chapter 32. Command Line Processor (CLP)

db2 - Command line processor invocation

The `db2` command starts the command line processor (CLP). The CLP is used to execute database utilities, SQL statements and online help. It offers a variety of command options, and can be started in:

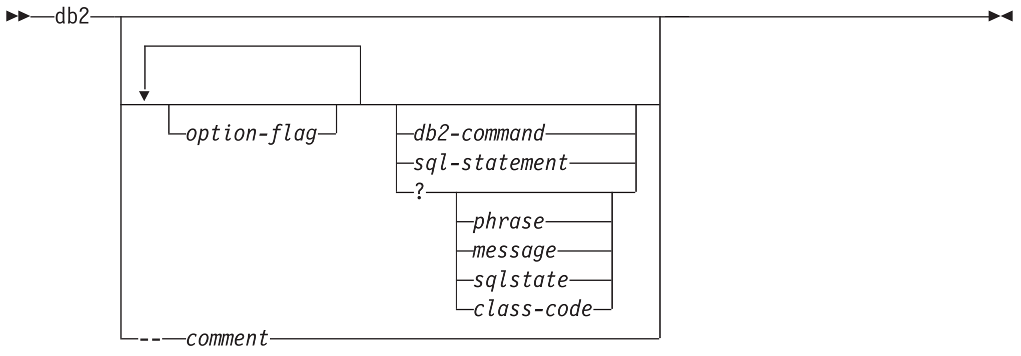
- Interactive input mode, characterized by the **`db2 =>`** input prompt
- Command mode, where each command must be prefixed by `db2`
- Batch mode, which uses the `-f` file input option.

On Windows operating systems, `db2cmd` opens the CLP-enabled DB2 window, and initializes the DB2 command line environment. Issuing this command is equivalent to clicking on the *DB2 Command Window* icon.

QUIT stops the command line processor. TERMINATE also stops the command line processor, but removes the associated back-end process and frees any memory that is being used. It is recommended that a TERMINATE be issued prior to every STOP DATABASE MANAGER (db2stop) command. It might also be necessary for a TERMINATE to be issued after database configuration parameters have been changed, in order for these changes to take effect. Existing connections should be reset before terminating the CLP.

The shell command (`!`), allows operating system commands to be executed from the interactive or the batch mode on UNIX based systems, and on Windows operating systems (`!ls` on UNIX, and `!dir` on Windows operating systems, for example).

Command Syntax:



option-flag
Specifies a CLP option flag.

db2-command
Specifies a DB2 command.

sql-statement
Specifies an SQL statement.

? Requests CLP general help.

? phrase
Requests the help text associated with a specified command or topic. If the database manager cannot find the requested information, it displays the general help screen.

`? options` requests a description and the current settings of the CLP options. `? help` requests information about reading the online help syntax diagrams.

? message
Requests help for a message specified by a valid SQLCODE (`? sql10007n`, for example).

? sqlstate
Requests help for a message specified by a valid SQLSTATE.

? class-code
Requests help for a message specified by a valid class-code.

-- comment
Input that begins with the comment characters `--` is treated as a comment by the command line processor.

In each case, a blank space must separate the question mark (?) from the variable name.

Related reference:
- "Command line processor options" on page 398
- "Command line processor return codes" on page 406
- "Command line processor features" on page 407

Command line processor options

The CLP command options can be specified by setting the command line processor DB2OPTIONS environment variable (which must be in uppercase), or with command line flags.

Users can set options for an entire session using DB2OPTIONS.

View the current settings for the option flags and the value of DB2OPTIONS using LIST COMMAND OPTIONS. Change an option setting from the interactive input mode or a command file using UPDATE COMMAND OPTIONS.

The command line processor sets options in the following order:
1. Sets up default options.
2. Reads DB2OPTIONS to override the defaults.
3. Reads the command line to override DB2OPTIONS.
4. Accepts input from UPDATE COMMAND OPTIONS as a final interactive override.

Table 117 on page 399 summarizes the CLP option flags. These options can be specified in any sequence and combination. To turn an option on, prefix the corresponding option letter with a minus sign (-). To turn an option off, either prefix the option letter with a minus sign and follow the option letter with another minus sign, or prefix the option letter with a plus sign (+). For example, `-c` turns the auto-commit option on, and either `-c-` or `+c` turns it off. These option letters are not case sensitive, that is, `-a` and `-A` are equivalent.

Table 117. CLP Command Options

Option Flag	Description	Default Setting
-a	This option tells the command line processor to display SQLCA data.	OFF
-c	This option tells the command line processor to automatically commit SQL statements.	ON
-d	This option tells the command line processor to retrieve and display XML declarations of XML data.	OFF
-e{c|s}	This option tells the command line processor to display SQLCODE or SQLSTATE. These options are mutually exclusive.	OFF
-f*filename*	This option tells the command line processor to read command input from a file instead of from standard input.	OFF
-i	This option tells the command line processor to 'pretty print' the XML data with proper indentation. This option will only affect the result set of XQuery statements.	OFF
-l*filename*	This option tells the command line processor to log commands in a history file.	OFF
-m	This option tells the command line processor to print the number of rows affected for `INSERT/DELETE/UPDATE/MERGE`.	OFF
-n	Removes the new line character within a single delimited token. If this option is not specified, the new line character is replaced with a space. This option must be used with the `-t` option.	OFF
-o	This option tells the command line processor to display output data and messages to standard output.	ON
-p	This option tells the command line processor to display a command line processor prompt when in interactive input mode.	ON
-q	This option tells the command line processor to preserve whitespaces and linefeeds in strings delimited with single or double quotes. When option `q` is `ON`, option `n` is ignored.	OFF
-r*filename*	This option tells the command line processor to write the report generated by a command to a file.	OFF
-s	This option tells the command line processor to stop execution if errors occur while executing commands in a batch file or in interactive mode.	OFF
-t	This option tells the command line processor to use a semicolon (;) as the statement termination character.	OFF
-td*x* or -td*xx*	This option tells the command line processor to define and to use *x* or *xx* as the statement termination character or characters (1 or 2 characters in length).	OFF
-v	This option tells the command line processor to echo command text to standard output.	OFF
-w	This option tells the command line processor to display FETCH/SELECT warning messages.	ON

Table 117. CLP Command Options (continued)

Option Flag	Description	Default Setting
-x	This option tells the command line processor to return data without any headers, including column names. This flag will not affect all commands. It applies to SQL statements and some commands that are based on SQL statements such as `LIST TABLES`.	OFF
-z*filename*	This option tells the command line processor to redirect all output to a file. It is similar to the `-r` option, but includes any messages or error codes with the output.	OFF

Example

The AIX command:

```
export DB2OPTIONS='+a -c +ec -o -p'
```

sets the following default settings for the session:

```
Display SQLCA   - off
Auto Commit     -  on
Display SQLCODE - off
Display Output  -  on
Display Prompt  -  on
```

The following is a detailed description of these options:

Show SQLCA Data Option (`-a`):
Displays SQLCA data to standard output after executing a DB2 command or an SQL statement. The SQLCA data is displayed instead of an error or success message.

The default setting for this command option is `OFF` (`+a` or `-a-`).

The `-o` and the `-r` options affect the `-a` option; see the option descriptions for details.

Auto-commit Option (`-c`):
This option specifies whether each command or statement is to be treated independently. If set `ON` (`-c`), each command or statement is automatically committed or rolled back. If the command or statement is successful, it and all successful commands and statements that were issued before it with autocommit `OFF` (`+c` or `-c-`) are committed. If, however, the command or statement fails, it and all successful commands and statements that were issued before it with autocommit `OFF` are rolled back. If set `OFF` (`+c` or `-c-`), `COMMIT` or `ROLLBACK` must be issued explicitly, or one of these actions will occur when the next command with autocommit `ON` (`-c`) is issued.

The default setting for this command option is `ON`.

The auto-commit option does not affect any other command line processor option.

Example: Consider the following scenario:

1. `db2 create database test`
2. `db2 connect to test`
3. `db2 +c "create table a (c1 int)"`
4. `db2 select c2 from a`

The SQL statement in step 4 fails because there is no column named C2 in table A. Since that statement was issued with auto-commit `ON` (default), it rolls back not only the statement in step 4, but also the one in step 3, because the latter was issued with auto-commit `OFF`. The command:

```
db2 list tables
```

then returns an empty list.

XML Declaration Option (`-d`):

The `-d` option tells the command line processor whether to retrieve and display XML declarations of XML data.

If set `ON` (`-d`), the XML declarations will be retrieved and displayed. If set `OFF` (`+d` or `-d-`), the XML declarations will not be retrieved and displayed. The default setting for this command option is `OFF`.

The XML declaration option does not affect any other command line processor options.

Display SQLCODE/SQLSTATE Option (`-e`):

The `-e{c|s}` option tells the command line processor to display the SQLCODE (`-ec`) or the SQLSTATE (`-es`) to standard output. Options `-ec` and `-es` are not valid in CLP interactive mode.

The default setting for this command option is `OFF` (`+e` or `-e-`).

The `-o` and the `-r` options affect the `-e` option; see the option descriptions for details.

The display SQLCODE/SQLSTATE option does not affect any other command line processor option.

Example: To retrieve SQLCODE from the command line processor running on AIX, enter:

```
sqlcode=`db2 –ec +o db2–command`
```

Read from Input File Option (`-f`):

The `-f`*filename* option tells the command line processor to read input from a specified file, instead of from standard input. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used.

When other options are combined with option `-f`, option `-f` must be specified last. For example:

```
db2 -tvf filename
```

This option cannot be changed from within the interactive mode.

The default setting for this command option is `OFF` (`+f` or `-f-`).

Commands are processed until the `QUIT` command or `TERMINATE` command is issued, or an end-of-file is encountered.

If both this option and a database command are specified, the command line processor does not process any commands, and an error message is returned.

Input file lines which begin with the comment characters `--` are treated as comments by the command line processor. Comment characters must be the first non-blank characters on a line.

Input file lines which begin with `(=` are treated as the beginning of a comment block. Lines which end with `=)` mark the end of a comment block. The block of input lines that begins at `(=` and ends at `=)` is treated as a continuous comment by the command line processor. Spaces before `(=` and after `=)` are allowed. Comments may be nested, and may be used nested in statements. The command termination character (`;`) cannot be used after `=)`.

If the `-f`*filename* option is specified, the `-p` option is ignored.

The read from input file option does not affect any other command line processor option.

Pretty Print Option (`-i`):

The `-i` option tells the command line processor to 'pretty print' the XML data with proper indentation. This option will only affect the result set of XQuery statements.

The default setting for this command option is `OFF` (`+i` or `-i-`).

The pretty print option does not affect any other command line processor options.

Log Commands in History File Option (-l):

The `-l`*filename* option tells the command line processor to log commands to a specified file. This history file contains records of the commands executed and their completion status. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used. If the specified file or default file already exists, the new log entry is appended to that file.

When other options are combined with option `-l`, option `-l` must be specified last. For example:

```
db2 -tvl filename
```

The default setting for this command option is `OFF` (`+l` or `-l-`).

The log commands in history file option does not affect any other command line processor option.

Display Number of Rows Affected Option (-m):

The `-m` option tells the command line processor whether or not to print the number of rows affected for `INSERT`, `DELETE`, `UPDATE`, or `MERGE`.

If set `ON (-m)`, the number of rows affected will be displayed for the statement of `INSERT/DELETE/UPDATE/MERGE`. If set `OFF (+m or -m-)`, the number of rows affected will not be displayed. For other statements, this option will be ignored. The default setting for this command option is `OFF`.

The `-o` and the `-r` options affect the `-m` option; see the option descriptions for details.

Remove New Line Character Option (-n):

Removes the new line character within a single delimited token. If this option is not specified, the new line character is replaced with a space. This option cannot be changed from within the interactive mode.

The default setting for this command option is `OFF` (`+n` or `-n-`).

This option must be used with the -t option; see the option description for details.

Display Output Option (-o):

The -o option tells the command line processor to send output data and messages to standard output.

The default setting for this command option is ON.

The interactive mode start-up information is not affected by this option. Output data consists of report output from the execution of the user-specified command, and SQLCA data (if requested).

The following options might be affected by the +o option:

- -r*filename*: Interactive mode start-up information is not saved.
- -e: SQLCODE or SQLSTATE is displayed on standard output even if +o is specified.
- -a: No effect if +o is specified. If -a, +o and -r*filename* are specified, SQLCA information is written to a file.

If both -o and -e options are specified, the data and either the SQLCODE or the SQLSTATE are displayed on the screen.

If both -o and -v options are specified, the data is displayed, and the text of each command issued is echoed to the screen.

The display output option does not affect any other command line processor option.

Display DB2 Interactive Prompt Option (-p):

The -p option tells the command line processor to display the command line processor prompt when the user is in interactive mode.

The default setting for this command option is ON.

Turning the prompt off is useful when commands are being piped to the command line processor. For example, a file containing CLP commands could be executed by issuing:

```
db2 +p < myfile.clp
```

The -p option is ignored if the -f*filename* option is specified.

The display DB2 interactive prompt option does not affect any other command line processor option.

Preserve Whitespaces and Linefeeds Option (-q):

The -q option tells the command line processor to preserve whitespaces and linefeeds in strings delimited with single or double quotes.

The default setting for this command option is OFF (+q or -q-).

If option -q is ON, option -n is ignored.

Save to Report File Option (-r):

The -r*filename* option causes any output data generated by a command to be written to a specified file, and is useful for capturing a report that would otherwise scroll off the screen. Messages or error codes are not written to the file. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used. New report entries are appended to the file.

The default setting for this command option is `OFF` (`+r` or `-r-`).

If the `-a` option is specified, SQLCA data is written to the file.

The `-r` option does not affect the `-e` option. If the `-e` option is specified, SQLCODE or SQLSTATE is written to standard output, not to a file.

If `-r`*filename* is set in DB2OPTIONS, the user can set the `+r` (or `-r-`) option from the command line to prevent output data for a particular command invocation from being written to the file.

The save to report file option does not affect any other command line processor option.

Stop Execution on Command Error Option (-s):

When commands are issued in interactive mode, or from an input file, and syntax or command errors occur, the `-s` option causes the command line processor to stop execution and to write error messages to standard output.

The default setting for this command option is `OFF` (`+s` or `-s-`). This setting causes the command line processor to display error messages, continue execution of the remaining commands, and to stop execution only if a system error occurs (return code 8).

The following table summarizes this behavior:

Table 118. CLP Return Codes and Command Execution

Return Code	`-s` Option Set	`+s` Option Set
0 (success)	execution continues	execution continues
1 (0 rows selected)	execution continues	execution continues
2 (warning)	execution continues	execution continues
4 (DB2 or SQL error)	execution stops	execution continues
8 (System error)	execution stops	execution stops

Statement Termination Character Option (-t):

The `-t` option tells the command line processor to use a semicolon (;) as the statement termination character, and disables the backslash (\) line continuation character. This option cannot be changed from within the interactive mode.

The default setting for this command option is `OFF` (`+t` or `-t-`).

To define termination characters 1 or 2 characters in length, use -td followed by the chosen character or characters. For example, `-td%%` sets `%%` as the statement termination characters. Alternatively, use the `--#SET TERMINATOR` directive to set the statement termination characters. For example, `--#SET TERMINATOR%%` sets `%%` as the statement termination characters.

The termination character cannot be used to concatenate multiple statements from the command line, since only the last non-blank character on each input line is checked for a termination symbol.

The statement termination character option does not affect any other command line processor option.

Verbose Output Option (-v):

The `-v` option causes the command line processor to echo (to standard output) the command text entered by the user prior to displaying the output, and any messages from that command. ECHO is exempt from this option.

The default setting for this command option is `OFF` (`+v` or `-v-`).

The `-v` option has no effect if `+o` (or `-o-`) is specified.

The verbose output option does not affect any other command line processor option.

Show Warning Messages Option (-w):

The `-w` option instructs the command line processor on whether or not to display warning messages that may occur during a query (FETCH/SELECT). Warnings can occur during various stages of the query execution which may result in the messages being displayed before, during or after the data is returned. To ensure the data returned does not contain warning message text this flag can be used.

The default setting for this command option is `ON`.

Suppress Printing of Column Headings Option (-x):

The `-x` option tells the command line processor to return data without any headers, including column names. This flag will not affect all commands. It applies to SQL statements and some commands that are based on SQL statements such as `LIST TABLES`.

The default setting for this command option is `OFF`.

Save all Output to File Option (-z):

The `-z`*filename* option causes all output generated by a command to be written to a specified file, and is useful for capturing a report that would otherwise scroll off the screen. It is similar to the `-r` option; in this case, however, messages, error codes, and other informational output are also written to the file. *Filename* is an absolute or relative file name which can include the directory path to the file. If the directory path is not specified, the current directory is used. New report entries are appended to the file.

The default setting for this command option is `OFF` (`+z` or `-z-`).

If the `-a` option is specified, SQLCA data is written to the file.

The `-z` option does not affect the `-e` option. If the `-e` option is specified, SQLCODE or SQLSTATE is written to standard output, not to a file.

If `-z`*filename* is set in DB2OPTIONS, the user can set the `+z` (or `-z-`) option from the command line to prevent output data for a particular command invocation from being written to the file.

The save all output to file option does not affect any other command line processor option.

Related reference:

- “db2 - Command line processor invocation” on page 397
- “Command line processor return codes” on page 406

Command line processor return codes

When the command line processor finishes processing a command or an SQL statement, it returns a return (or exit) code. These codes are transparent to users executing CLP functions from the command line, but they can be retrieved when those functions are executed from a shell script.

For example, the following Bourne shell script executes the GET DATABASE MANAGER CONFIGURATION command, then inspects the CLP return code:

```
db2 get database manager configuration
if [ "$?" = "0" ]
then echo "OK!"
fi
```

The return code can be one of the following:

Code	**Description**
0	DB2 command or SQL statement executed successfully
1	SELECT or FETCH statement returned no rows
2	DB2 command or SQL statement warning
4	DB2 command or SQL statement error
8	Command line processor system error

The command line processor does not provide a return code while a user is executing statements from interactive mode, or while input is being read from a file (using the `-f` option).

A return code is available only after the user quits interactive mode, or when processing of an input file ends. In these cases, the return code is the logical OR of the distinct codes returned from the individual commands or statements executed to that point.

For example, if a user in interactive mode issues commands resulting in return codes of 0, 1, and 2, a return code of 3 will be returned after the user quits interactive mode. The individual codes 0, 1, and 2 are not returned. Return code 3 tells the user that during interactive mode processing, one or more commands returned a 1, and one or more commands returned a 2.

A return code of 4 results from a negative SQLCODE returned by a DB2 command or an SQL statement. A return code of 8 results only if the command line processor encounters a system error.

If commands are issued from an input file or in interactive mode, and the command line processor experiences a system error (return code 8), command execution is halted immediately. If one or more DB2 commands or SQL statements end in error (return code 4), command execution stops if the `-s` (Stop Execution on Command Error) option is set; otherwise, execution continues.

Related reference:

- "db2 - Command line processor invocation" on page 397
- "Command line processor options" on page 398

Command line processor features

The command line processor operates as follows:

- The CLP command (in either case) is typed at the command prompt.
- The command is sent to the command shell by pressing the ENTER key.
- Output is automatically directed to the standard output device.
- Piping and redirection are supported.
- The user is notified of successful and unsuccessful completion.
- Following execution of the command, control returns to the operating system command prompt, and the user can enter more commands.

Certain CLP commands and SQL statements require that the server instance is running and a database connection exists. Connect to a database by doing one of the following:

- Issue the SQL statement `DB2 CONNECT TO` *`database`*.
- Establish an implicit connection to the default database defined by the DB2 registry variable DB2DBDFT.

If a command exceeds the character limit allowed at the command prompt, a backslash (\) can be used as the line continuation character. When the command line processor encounters the line continuation character, it reads the next line and concatenates the characters contained on both lines. Alternatively, the `-t` option can be used to set a different line termination character.

The command line processor recognizes a string called `NULL` as a null string. Fields that have been set previously to some value can later be set to NULL. For example,

```
db2 update database manager configuration using tm_database NULL
```

sets the *tm_database* field to NULL. This operation is case sensitive. A lowercase `null` is not interpreted as a null string, but rather as a string containing the letters `null`.

Customizing the Command Line Processor:

It is possible to customize the interactive input prompt by using the DB2_CLPPROMPT registry variable. This registry variable can be set to any text string of maximum length 100 and can contain the tokens %i, %ia, %d, %da and %n. Specific values will be substituted for these tokens at run-time.

Table 119. DB2_CLPPROMPT tokens and run-time values

DB2_CLPPROMPT token	Value at run-time
%ia	Authorization ID of the current instance attachment
%i	Local alias of the currently attached instance. If no instance attachment exists, the value of the DB2INSTANCE registry variable. On Windows platforms only, if the DB2INSTANCE registry variable is not set, the value of the DB2INSTDEF registry variable.
%da	Authorization ID of the current database connection
%d	Local alias of the currently connected database. If no database connection exists, the value of the DB2DBDFT registry variable.
%n	New line

- If any token has no associated value at run-time, the empty string is substituted for that token.
- The interactive input prompt will always present the authorization IDs, database names, and instance names in upper case, so as to be consistent with the connection and attachment information displayed at the prompt.
- If the DB2_CLPPROMPT registry variable is changed within CLP interactive mode, the new value of DB2_CLPPROMPT will not take effect until CLP interactive mode has been closed and reopened.

Examples:

If DB2_CLPPROMPT is defined as `(%ia@%i, %da@%d)`, the input prompt will have the following values:

- No instance attachment and no database connection. DB2INSTANCE set to ″DB2″. DB2DBDFT is not set.
  ```
  (@DB2, @)
  ```
- (Windows) No instance attachment and no database connection. DB2INSTANCE and DB2DBDFT not set. DB2INSTDEF set to ″DB2″.
  ```
  (@DB2, @)
  ```
- No instance attachment and no database connection. DB2INSTANCE set to ″DB2″. DB2DBDFT set to ″SAMPLE″.
  ```
  (@DB2, @SAMPLE)
  ```
- Instance attachment to instance ″DB2″ with authorization ID ″tyronnem″. DB2INSTANCE set to ″DB2″. DB2DBDFT set to ″SAMPLE″.
  ```
  (TYRONNEM@DB2, @SAMPLE)
  ```
- Database connection to database ″sample″ with authorization ID ″horman″. DB2INSTANCE set to ″DB2″. DB2DBDFT set to ″SAMPLE″.
  ```
  (@DB2, HORMAN@SAMPLE)
  ```
- Instance attachment to instance ″DB2″ with authorization ID ″tyronnem″. Database connection to database ″sample″ with authorization ID ″horman″. DB2INSTANCE set to ″DB2″. DB2DBDFT not set.
  ```
  (TYRONNEM@DB2, HORMAN@SAMPLE)
  ```

Using the Command Line Processor in Command Files:

CLP requests to the database manager can be imbedded in a shell script command file. The following example shows how to enter the CREATE TABLE statement in a shell script command file:

```
   db2 "create table mytable (name VARCHAR(20), color CHAR(10))"
```

For more information about commands and command files, see the appropriate operating system manual.

Command Line Processor Design:

The command line processor consists of two processes: the front-end process (the DB2 command), which acts as the user interface, and the back-end process (db2bp), which maintains a database connection.

Maintaining Database Connections

Each time that db2 is invoked, a new front-end process is started. The back-end process is started by the first db2 invocation, and can be explicitly terminated with TERMINATE. All front-end processes with the same parent are serviced by a single back-end process, and therefore share a single database connection.

For example, the following db2 calls from the same operating system command prompt result in separate front-end processes sharing a single back-end process, which holds a database connection throughout:

- db2 'connect to sample',
- db2 'select * from org',
- . foo (where foo is a shell script containing DB2 commands), and
- db2 -tf myfile.clp.

The following invocations from the same operating system prompt result in separate database connections because each has a distinct parent process, and therefore a distinct back-end process:

- foo
- . foo &
- foo &
- sh foo

Communication between Front-end and Back-end Processes

The front-end process and back-end processes communicate through three message queues: a request queue, an input queue, and an output queue.

Environment Variables

The following environment variables offer a means of configuring communication between the two processes:

Table 120. Environment Variables

Variable	Minimum	Maximum	Default
DB2BQTIME	1 second	5294967295	1 second
DB2BQTRY	0 tries	5294967295	60 tries
DB2RQTIME	1 second	5294967295	5 seconds
DB2IQTIME	1 second	5294967295	5 seconds

DB2BQTIME
When the command line processor is invoked, the front-end process checks if the back-end process is already active. If it is active, the front-end process reestablishes a connection to it. If it is not active, the front-end process activates it. The front-end process then idles for the duration specified by the DB2BQTIME variable, and checks again. The front-end process continues to check for the number of times specified by the DB2BQTRY variable, after which, if the back-end process is still not active, it times out and returns an error message.

DB2BQTRY
Works in conjunction with the DB2BQTIME variable, and specifies the number of times the front-end process tries to determine whether the back-end process is active.

The values of DB2BQTIME and DB2BQTRY can be increased during peak periods to optimize query time.

DB2RQTIME
Once the back-end process has been started, it waits on its request queue for a request from the front-end. It also waits on the request queue between requests initiated from the command prompt.

The DB2RQTIME variable specifies the length of time the back-end process waits for a request from the front-end process. At the end of this time, if no request is present on the request queue, the back-end process checks whether the parent of the front-end process still exists, and terminates itself if it does not exist. Otherwise, it continues to wait on the request queue.

DB2IQTIME
When the back-end process receives a request from the front-end process, it sends an acknowledgment to the front-end process indicating that it is ready to receive input via the input queue. The back-end process then waits on its input queue. It also waits on the input queue while a batch file (specified with the `-f` option) is executing, and while the user is in interactive mode.

The DB2IQTIME variable specifies the length of time the back-end process waits on the input queue for the front-end process to pass the commands. After this time has elapsed, the back-end process checks whether the front-end process is active, and returns to wait on the request queue if the front-end process no longer exists. Otherwise, the back-end process continues to wait for input from the front-end process.

To view the values of these environment variables, use LIST COMMAND OPTIONS.

The back-end environment variables inherit the values set by the front-end process at the time the back-end process is initiated. However, if the front-end environment variables are changed, the back-end process will not inherit these changes. The back-end process must first be terminated, and then restarted (by issuing the `db2` command) to inherit the changed values.

An example of when the back-end process must be terminated is provided by the following scenario:

1. User A logs on, issues some CLP commands, and then logs off without issuing TERMINATE.
2. User B logs on using the same window.
3. When user B issues certain CLP commands, they fail with message DB21016 (system error).

The back-end process started by user A is still active when user B starts using the CLP, because the parent of user B's front-end process (the operating system window from which the commands are issued) is still active. The back-end process attempts to service the new commands issued by user B; however, user B's front-end process does not have enough authority to use the message queues of the back-end process, because it needs the authority of user A, who created that back-end process. A CLP session must end with a TERMINATE command before a user starts a new CLP session using the same operating system window. This creates a fresh back-end process for each new user, preventing authority problems, and setting the correct values of environment variables (such as DB2INSTANCE) in the new user's back-end process.

CLP Usage Notes:

Commands can be entered either in upper case or in lowercase from the command prompt. However, parameters that are case sensitive to DB2 must be entered in the exact case desired. For example, the *comment-string* in the WITH clause of the CHANGE DATABASE COMMENT command is a case sensitive parameter.

Delimited identifiers are allowed in SQL statements.

Special characters, or metacharacters (such as $ & * ( ) ; < > ? \ ' ") are allowed within CLP commands. If they are used outside the CLP interactive mode, or the CLP batch input mode, these characters are interpreted by the operating system shell. Quotation marks or an escape character are required if the shell is not to take any special action.

For example, when executed inside an AIX Korn shell environment,

```
db2 select * from org where division > 'Eastern'
```

is interpreted as ″select <the names of all files> from org where division″. The result, an SQL syntax error, is redirected to the file `Eastern`. The following syntax produces the correct output:

```
db2 "select * from org where division > 'Eastern'"
```

Special characters vary from platform to platform. In the AIX Korn shell, the above example could be rewritten using an escape character (\), such as *, \>, or \'.

Most operating system environments allow input and output to be redirected. For example, if a connection to the SAMPLE database has been made, the following request queries the STAFF table, and sends the output to a file named `staflist.txt` in the `mydata` directory:

```
db2 "select * from staff" > mydata/staflist.txt
```

For environments where output redirection is not supported, CLP options can be used. For example, the request can be rewritten as

```
db2 -r mydata\staflist.txt "select * from staff"

db2 -z mydata\staflist.txt "select * from staff"
```

The command line processor is not a programming language. For example, it does not support host variables, and the statement,

```
db2 connect to :HostVar in share mode
```

is syntactically incorrect, because `:HostVar` is not a valid database name.

The command line processor represents SQL NULL values as hyphens (-). If the column is numeric, the hyphen is placed at the right of the column. If the column is not numeric, the hyphen is at the left.

To correctly display the national characters for single byte (SBCS) languages from the DB2 command line processor window, a True Type font must be selected. For example, in a Windows environment, open the command window properties notebook and select a font such as Lucinda Console.

Related concepts:

- “DB2 Command Line Processor (CLP)” in *Developing SQL and External Routines*

Command line processor features

Related reference:

- "Command line processor options" on page 398
- "Command line processor return codes" on page 406

Chapter 33. Security considerations for utilities

Privileges, authorities and authorization required to use export

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM or DBADM authority, or CONTROL or SELECT privilege for each table participating in the export operation.

Related reference:

- "db2Export - Export data from a database" on page 684
- "EXPORT " on page 496

Privileges, authorities, and authorization required to use backup

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the backup utility.

Related reference:

- "BACKUP DATABASE " on page 437
- "db2Backup - Back up a database or table space" on page 665

Privileges, authorities, and authorization required to use recover

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the recover utility.

Related reference:
- "db2Recover - Restore and roll forward a database" on page 743
- "RECOVER DATABASE" on page 571

Privileges, authorities, and authorization required to use restore

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to restore to an *existing* database from a full database backup. To restore to a *new* database, you must have SYSADM or SYSCTRL authority.

Related reference:
- "db2Restore - Restore a database or table space" on page 756
- "RESTORE DATABASE " on page 591

Privileges, authorities, and authorization required to use rollforward

Privileges enable users to create or access database resources. Authority levels provide a method of grouping privileges and higher-level database manager maintenance and utility operations. Together, these act to control access to the database manager and its database objects. Users can access only those objects for which they have the appropriate authorization; that is, the required privilege or authority.

You must have SYSADM, SYSCTRL, or SYSMAINT authority to use the rollforward utility.

Related reference:
- "db2Rollforward - Roll forward a database" on page 767
- "ROLLFORWARD DATABASE " on page 606

Privileges, authorities, and authorizations required to use Load

To load data into a table, you must have one of the following:
- SYSADM authority
- DBADM authority
- LOAD authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)

– INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
– INSERT privilege on the exception table, if such a table is used as part of the load operation.

Since all load processes (and all DB2 server processes, in general), are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

On Windows, and Windows.NET operating systems where DB2 is running as a Windows service, if you are loading data from files that reside on a network drive, you must configure the DB2 service to run under a user account that has read access to these files.

Notes:
- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table.
- To load data into a table that has protected rows, the session authorization ID must have been granted a security label for write access that is part of the security policy protecting the table.

Related reference:
- "db2Load - Load data into a table" on page 723
- "LOAD " on page 542

Chapter 34. Issuing commands to multiple database partitions

This section describes how to issue commands to multiple database partitions, including problem resolution.

Issuing commands in a partitioned database environment

In a partitioned database environment, you might want to issue commands to be run on computers in the instance, or on database partition servers (nodes). You can do so using the `rah` command or the `db2_all` command. The `rah` command allows you to issue commands that you want to run at computers in the instance. If you want the commands to run at database partition servers in the instance, you run the `db2_all` command. This section provides an overview of these commands. The information that follows applies to partitioned database environments only.

Notes:

1. On Linux and UNIX platforms, your login shell can be a Korn shell or any other shell; however, there are differences in the way the different shells handle commands containing special characters.
2. Also, on Linux and UNIX platforms, rah uses the remote shell program specified by the DB2RSHCMD registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). The ssh remote shell program is used to prevent the transmission of passwords in clear text in UNIX operating system environments. If this registry variable is not set, rsh (or remsh for HP-UX) is used.
3. On Windows, to run the `rah` command or the `db2_all` command, you must be logged on with a user account that is a member of the Administrators group.

To determine the scope of a command, refer to the *Command Reference*, which indicates whether a command runs on a single database partition server, or on all of them. If the command runs on one database partition server and you want it to run on all of them, use `db2_all`. The exception is the `db2trc` command, which runs on all the logical nodes (database partition servers) on a computer. If you want to run `db2trc` on all logical nodes on all computers, use `rah`.

Related concepts:

- "rah and db2_all commands overview" on page 417
- "Specifying the rah and db2_all commands" on page 419

Related reference:

- "rah and db2_all command descriptions" on page 418

rah and db2_all commands overview

You can run the commands sequentially at one database partition server after another, or you can run the commands in parallel. On Linux and UNIX platforms, if you run the commands in parallel, you can either choose to have the output sent to a buffer and collected for display (the default behavior) or the output can be displayed at the computer where the command is issued. On Windows, if you run the commands in parallel, the output is displayed at the computer where the command is issued.

To use the `rah` command, type:

```
rah command
```

To use the `db2_all` command, type:

```
db2_all command
```

To obtain help about `rah` syntax, type

```
rah "?"
```

The command can be almost anything which you could type at an interactive prompt, including, for example, multiple commands to be run in sequence. On Linux and UNIX platforms, you separate multiple commands using a semicolon (;). On Windows, you separate multiple commands using an ampersand (&). Do not use the separator character following the last command.

The following example shows how to use the `db2_all` command to change the database configuration on all database partitions that are specified in the node configuration file. Because the ; character is placed inside double quotation marks, the request will run concurrently:

```
db2_all ";DB2 UPDATE DB CFG FOR sample USING LOGFILSIZ 100"
```

Related concepts:

- "Issuing commands in a partitioned database environment" on page 417
- "Specifying the rah and db2_all commands" on page 419

Related reference:

- "rah and db2_all command descriptions" on page 418

rah and db2_all command descriptions

You can use the following commands:

Command	Description
rah	Runs the command on all computers.
db2_all	Runs the command on all database partition servers that you specify.
db2_kill	Abruptly stops all processes being run on multiple database partition servers and cleans up all resources on all database partition servers. This command renders your databases inconsistent. Do *not* issue this command except under direction from IBM service.
db2_call_stack	On Linux and UNIX platforms, causes all processes running on all database partition servers to write call traceback to the syslog. On Windows, causes all processes running on all database partition servers to write call traceback to the P*xxxx.nnn* file in the instance directory, where P*xxxx* is the process ID and *nnn* is the database partition number.

On Linux and UNIX platforms, these commands execute `rah` with certain implicit settings such as:

- Run in parallel at all computers
- Buffer command output in /tmp/$USER/db2_kill, /tmp/$USER/db2_call_stack respectively.

On Windows, these commands execute `rah` to run in parallel at all computers.

Related concepts:

- "rah and db2_all commands overview" on page 417
- "Running commands in parallel on Linux and UNIX platforms" on page 420
- "Specifying the rah and db2_all commands" on page 419

Specifying the rah and db2_all commands

You can specify the command:

- From the command line as the parameter
- In response to the prompt if you don't specify any parameter.

You should use the prompt method if the command contains the following special characters:

```
| & ; < > ( )  { } [ ] unsubstituted $
```

If you specify the command as the parameter on the command line, you must enclose it in double quotation marks if it contains any of the special characters just listed.

Note: On Linux and UNIX platforms, the command will be added to your command history just as if you typed it at the prompt.

All special characters in the command can be entered normally (without being enclosed in quotation marks, except for \). If you need to include a \ in your command, you must type two backslashes (\\).

Note: On Linux and UNIX platforms, if you are not using a Korn shell, all special characters in the command can be entered normally (without being enclosed in quotation marks, except for ", \, unsubstituted $, and the single quotation mark (')). If you need to include one of these characters in your command, you must precede them by three backslashes (\\\). For example, if you need to include a \ in your command, you must type four backslashes (\\\\).

If you need to include a double quotation mark (") in your command, you must precede it by three backslashes, for example, \\\".

Notes:

1. On Linux and UNIX platforms, you cannot include a single quotation mark (') in your command unless your command shell provides some way of entering a single quotation mark inside a singly quoted string.
2. On Windows, you cannot include a single quotation mark (') in your command unless your command window provides some way of entering a single quotation mark inside a singly quoted string.

When you run any korn-shell shell-script which contains logic to read from stdin in the background, you should explicitly redirect stdin to a source where the process can read without getting stopped on the terminal (SIGTTIN message). To redirect stdin, you can run a script with the following form:

```
shell_script </dev/null &
```

if there is no input to be supplied.

In a similar way, you should always specify `</dev/null` when running db2_all in the background. For example:

```
db2_all ";run_this_command" </dev/null &
```

By doing this you can redirect stdin and avoid getting stopped on the terminal.

An alternative to this method, when you are not concerned about output from the remote command, is to use the "daemonize" option in the db2_all prefix:

```
db2_all ";daemonize_this_command" &
```

Related concepts:

- "Additional rah information (Solaris and AIX only)" on page 422
- "Running commands in parallel on Linux and UNIX platforms" on page 420

Related tasks:

- "Setting the default environment profile for rah on Windows" on page 428

Related reference:

- "Controlling the rah command" on page 426
- "rah and db2_all command descriptions" on page 418
- "rah command prefix sequences" on page 422

Running commands in parallel on Linux and UNIX platforms

Note: The information in this section applies to Linux and UNIX platforms only.

By default, the command is run sequentially at each computer, but you can specify to run the commands in parallel using background rshells by prefixing the command with certain prefix sequences. If the rshell is run in the background, then each command puts the output in a buffer file at its remote computer. This process retrieves the output in two pieces:

1. After the remote command completes.
2. After the rshell terminates, which might be later if some processes are still running.

The name of the buffer file is /tmp/$USER/rahout by default, but it can be specified by the environment variables $RAHBUFDIR/$RAHBUFNAME.

When you specify that you want the commands to be run concurrently, by default, this script prefixes an additional command to the command sent to all hosts to check that $RAHBUFDIR and $RAHBUFNAME are usable for the buffer file. It creates $RAHBUFDIR. To suppress this, export an environment variable `RAHCHECKBUF=no`. You can do this to save time if you know the directory exists and is usable.

Before using rah to run a command concurrently at multiple computers:

- Ensure that a directory /tmp/$USER exists for your user ID at each computer. To create a directory if one does not already exist, run:
```
rah ")mkdir /tmp/$USER"
```
- Add the following line to your .kshrc (for Korn shell syntax) or .profile, and also type it into your current session:
```
export RAHCHECKBUF=no
```
- Ensure that each computer ID at which you run the remote command has an entry in its .rhosts file for the ID which runs rah; and the ID which runs rah has an entry in its .rhosts file for each computer ID at which you run the remote command.

Related concepts:

- "Additional rah information (Solaris and AIX only)" on page 422

Related tasks:

- "Monitoring rah processes on Linux and UNIX platforms" on page 421

Related reference:

- "Determining problems with rah on Linux and UNIX platforms" on page 428
- "rah command prefix sequences" on page 422

Monitoring rah processes on Linux and UNIX platforms

Procedure:

Note: The information in this section applies to Linux and UNIX platforms only. While any remote commands are still running or buffered output is still being accumulated, processes started by rah monitor activity to:

- Write messages to the terminal indicating which commands have not been run
- Retrieve buffered output.

The informative messages are written at an interval controlled by the environment variable RAHWAITTIME. Refer to the help information for details on how specify this. All informative messages can be completely suppressed by exporting RAHWAITTIME=0.

The primary monitoring process is a command whose command name (as shown by the ps command) is rahwaitfor. The first informative message tells you the pid (process id) of this process. All other monitoring processes will appear as ksh commands running the rah script (or the name of the symbolic link). If you want, you can stop all monitoring processes by the command:
```
kill <pid>
```

where <pid> is the process ID of the primary monitoring process. Do not specify a signal number. Leave the default of 15. This will not affect the remote commands at all, but will prevent the automatic display of buffered output. Note that there might be two or more different sets of monitoring processes executing at different times during the life of a single execution of rah. However, if at any time you stop the current set, then no more will be started.

If your regular login shell is not a Korn shell (for example /bin/ksh), you can use `rah`, but there are some slightly different rules on how to enter commands containing the following special characters:

```
  "  unsubstituted $ '
```

For more information, type `rah "?"`. Also, in a Linux and UNIX environment, if the login shell at the ID which executes the remote commands is not a Korn shell, then the login shell at the ID which executes `rah` must also not be a Korn shell. (`rah` makes the decision as to whether the remote ID's shell is a Korn shell based on the local ID). The shell must not perform any substitution or special processing on a string enclosed in single quotation marks. It must leave it exactly as is.

Related concepts:
- "Additional rah information (Solaris and AIX only)" on page 422
- "Running commands in parallel on Linux and UNIX platforms" on page 420

Additional rah information (Solaris and AIX only)

To enhance performance, rah has been extended to use tree_logic on large systems. That is, rah will check how many nodes the list contains, and if that number exceeds a threshold value, it constructs a subset of the list and sends a recursive invocation of itself to those nodes. At those nodes, the recursively invoked rah follows the same logic until the list is small enough to follow the standard logic (now the ″leaf-of-tree″ logic) of sending the command to all nodes on the list. The threshold can be specified by environment variable RAHTREETHRESH, or defaults to 15.

In the case of a multiple-logical-node-per-physical-node system, db2_all will favor sending the recursive invocation to distinct physical nodes, which will then rsh to other logical nodes on the same physical node, thus also reducing inter-physical-node traffic. (This point applies only to db2_all, not rah, since rah always sends only to distinct physical nodes.)

Related concepts:
- "Running commands in parallel on Linux and UNIX platforms" on page 420

Related tasks:
- "Monitoring rah processes on Linux and UNIX platforms" on page 421

rah command prefix sequences

A prefix sequence is one or more special characters. Type one or more prefix sequences immediately preceding the characters of the command without any intervening blanks. If you want to specify more than one sequence, you can type them in any order, but characters within any multicharacter sequence must be typed in order. If you type any prefix sequences, you must enclose the entire command, including the prefix sequences in double quotation marks, as in the following examples:
- On Linux and UNIX platforms:
  ```
  rah "};ps -F pid,ppid,etime,args -u $USER"
  ```
- On Windows:
  ```
  rah "||db2 get db cfg for sample"
  ```

The prefix sequences are:

Sequence	**Purpose**
`|`	Runs the commands in sequence in the background.
`|`&	Runs the commands in sequence in the background and terminates the command after all remote commands have completed, even if some processes are still running. This might be later if, for example, child processes (on Linux and UNIX platforms) or background processes (on Windows) are still running. In this case, the command starts a separate background process to retrieve any remote output generated after command termination and writes it back to the originating computer. **Note:** On Linux and UNIX platforms, specifying `&` degrades performance, because more `rsh` commands are required.
`||`	Runs the commands in parallel in the background.
`||`&	Runs the commands in parallel in the background and terminates the command after all remote commands have completed as described for the `|`& case above. **Note:** On Linux and UNIX platforms, specifying `&` degrades performance, because more `rsh` commands are required.
;	Same as `||`& above. This is an alternative shorter form. **Note:** On Linux and UNIX platforms, specifying ; degrades performance relative to `||`, because more `rsh` commands are required.
]	Prepends dot-execution of user's profile before executing command. **Note:** Available on Linux and UNIX platforms only.
}	Prepends dot-execution of file named in $RAHENV (probably .kshrc) before executing command. **Note:** Available on Linux and UNIX platforms only.
]}	Prepends dot-execution of user's profile followed by execution of file named in $RAHENV (probably .kshrc) before executing command. **Note:** Available on Linux and UNIX platforms only.
)	Suppresses execution of user's profile and of file named in $RAHENV. **Note:** Available on Linux and UNIX platforms only.
'	Echoes the command invocation to the computer.
<	Sends to all the computers except this one.
<<–nnn<	Sends to all-but-database partition server *nnn* (all database partition servers in db2nodes.cfg except for node number *nnn*, see the first paragraph following the last prefix sequence in this table).
<<+nnn<	Sends to only database partition server *nnn* (the database partition

server in db2nodes.cfg whose database partition number is *nnn*, see the first paragraph following the last prefix sequence in this table).

(blank character)
Runs the remote command in the background with `stdin`, `stdout`, and `stderr` all closed. This option is valid only when running the command in the background, that is, only in a prefix sequence which also includes \ or ;. It allows the command to complete much sooner (as soon as the remote command has been initiated). If you specify this prefix sequence on the `rah` command line, then either enclose the command in single quotation marks, or enclose the command in double quotation marks, and precede the prefix character by \ . For example,

```
rah '; mydaemon'
```

or

```
rah ";\ mydaemon"
```

When run as a background process, the `rah` command will never wait for any output to be returned.

> Substitutes occurrences of <> with the computer name.

" Substitutes occurrences of () by the computer index, and substitutes occurrences of ## by the database partition number.

Notes:

1. The computer index is a number that associated with a computer in the database system. If you are not running multiple logical partitions, the computer index for a computer corresponds to the database partition number for that computer in the node configuration file. To obtain the computer index for a computer in a multiple logical partition database environment, do not count duplicate entries for those computers that run multiple logical partitions. For example, if MACH1 is running two logical partitions and MACH2 is also running two logical partitions, the database partition number for MACH3 is `5` in the node configuration file. The computer index for MACH3, however, would be 3.

   On Windows, do not edit the node configuration file. To obtain the computer index, use the `db2nlist` command.
2. When " is specified, duplicates are not eliminated from the list of computers.

When using the `<<-nnn<` and `<<+nnn<` prefix sequences, *nnn* is any 1-, 2- or 3-digit database partition number which must match the *nodenum* value in the db2nodes.cfg file.

Note: Prefix sequences are considered to be part of the command. If you specify a prefix sequence as part of a command, you must enclose the entire command, including the prefix sequences, in double quotation marks.

Related concepts:

- "Running commands in parallel on Linux and UNIX platforms" on page 420
- "Specifying the rah and db2_all commands" on page 419

Related reference:

- "rah and db2_all command descriptions" on page 418

Specifying the list of computers in a partitioned database environment

Procedure:

By default, the list of computers is taken from the node configuration file, db2nodes.cfg. You can override this by:

- Specifying a pathname to the file that contains the list of computers by exporting (on Linux and UNIX platforms) or setting (on Windows) the environment variable RAHOSTFILE.
- Specifying the list explicitly, as a string of names separated by spaces, by exporting (on Linux and UNIX platforms) or setting (on Windows) the environment variable RAHOSTLIST.

  Note: If both of these environment variables are specified, RAHOSTLIST takes precedence.

Note: On Windows, to avoid introducing inconsistencies into the node configuration file, do *not* edit it manually. To obtain the list of computers in the instance, use the `db2nlist` command.

Related tasks:

- "Eliminating duplicate entries from a list of computers in a partitioned database environment" on page 425

Eliminating duplicate entries from a list of computers in a partitioned database environment

Procedure:

If you are running DB2 Enterprise Server Edition with multiple logical database partition servers on one computer, your db2nodes.cfg file will contain multiple entries for that computer. In this situation, the `rah` command needs to know whether you want the command to be executed once only on each computer or once for each logical database partition listed in the db2nodes.cfg file. Use the `rah` command to specify computers. Use the `db2_all` command to specify logical database partitions.

Note: On Linux and UNIX platforms, if you specify computers, `rah` will normally eliminate duplicates from the computer list, with the following exception: if you specify logical database partitions, `db2_all` prepends the following assignment to your command:

```
export DB2NODE=nnn  (for Korn shell syntax)
```

where *nnn* is the database partition number taken from the corresponding line in the db2nodes.cfg file, so that the command will be routed to the desired database partition server.

When specifying logical database partitions, you can restrict the list to include all logical database partitions except one, or only specify one database partition server using the `<<-nnn<` and `<<+nnn<` prefix sequences. You might want to do this if you want to run a command to catalog the database partition first, and when that has completed, run the same command at all other database partition servers, possibly in parallel. This is usually required when running the `db2 restart database` command. You will need to know the database partition number of the catalog partition to do this.

If you execute `db2 restart database` using the `rah` command, duplicate entries are eliminated from the list of computers. However if you specify the ″ prefix, then duplicates are not eliminated, because it is assumed that use of the ″ prefix implies sending to each database partition server, rather than to each computer.

Related tasks:

- "Specifying the list of computers in a partitioned database environment" on page 425

Related reference:

- "RESTART DATABASE " on page 589
- "rah command prefix sequences" on page 422

Controlling the rah command

You can use the following environment variables to control the `rah` command.

Table 121.

Name	Meaning	Default
$RAHBUFDIR **Note:** Available on Linux and UNIX platforms only.	Directory for buffer	/tmp/$USER
$RAHBUFNAME **Note:** Available on Linux and UNIX platforms only.	Filename for buffer	rahout
$RAHOSTFILE (on Linux and UNIX platforms); RAHOSTFILE (on Windows)	File containing list of hosts	db2nodes.cfg
$RAHOSTLIST (on Linux and UNIX platforms); RAHOSTLIST (on Windows)	List of hosts as a string	extracted from $RAHOSTFILE
$RAHCHECKBUF **Note:** Available on Linux and UNIX platforms only.	If set to ″no″, bypass checks	not set
$RAHSLEEPTIME (on Linux and UNIX platforms); RAHSLEEPTIME (on Windows)	Time in seconds this script will wait for initial output from commands run in parallel	86400 seconds for `db2_kill`, 200 seconds for all other

Table 121. (continued)

Name	Meaning	Default
$RAHWAITTIME (on Linux and UNIX platforms); RAHWAITTIME (on Windows)	On Windows, interval in seconds between successive checks that remote jobs are still running. On Linux and UNIX platforms, interval in seconds between successive checks that remote jobs are still running and `rah: waiting for <pid> ...` messages. On all platforms, specify any positive integer. Prefix value with a leading zero to suppress messages, for example, export RAHWAITTIME=045. It is not necessary to specify a low value as `rah` does not rely on these checks to detect job completion.	45 seconds
$RAHENV **Note:** Available on Linux and UNIX platforms only.	Specifies filename to be executed if $RAHDOTFILES=E or K or PE or B	$ENV
$RAHUSER (on Linux and UNIX platforms); RAHUSER (on Windows)	On Linux and UNIX platforms, user ID under which the remote command is to be run. On Windows, the logon account associated with the DB2 Remote Command Service	$USER

Note: On Linux and UNIX platforms, the value of $RAHENV where `rah` is run is used, not the value (if any) set by the remote shell.

Related reference:

- "Using $RAHDOTFILES on Linux and UNIX platforms" on page 427

Using $RAHDOTFILES on Linux and UNIX platforms

Note: The information in this section applies to Linux and UNIX platforms only.

Following are the .files that are run if no prefix sequence is specified:

P	.profile
E	File named in $RAHENV (probably .kshrc)
K	Same as E
PE	.profile followed by file named in $RAHENV (probably .kshrc)
B	Same as PE
N	None (or Neither)

Note: If your login shell is not a Korn shell, any dot files you specify to be executed will be executed in a Korn shell process, and so must conform to Korn shell syntax. So, for example, if your login shell is a C shell, to have your .cshrc environment set up for commands executed by `rah`, you should either create a Korn shell INSTHOME/.profile equivalent to your .cshrc and specify in your INSTHOME/.cshrc:

```
setenv RAHDOTFILES P
```

or you should create a Korn shell INSTHOME/.kshrc equivalent to your .cshrc and specify in your INSTHOME/.cshrc:

```
setenv RAHDOTFILES E
setenv RAHENV INSTHOME/.kshrc
```

Also, it is essential that your .cshrc does not write to stdout if there is no tty (as when invoked by `rsh`). You can ensure this by enclosing any lines which write to stdout by, for example,

```
if { tty -s } then echo "executed .cshrc";
endif
```

Related reference:

- "Controlling the rah command" on page 426

Setting the default environment profile for rah on Windows

Procedure:

Note: The information in this section applies to Windows® only.
To set the default environment profile for the `rah` command, use a file called db2rah.env, which should be created in the instance directory. The file should have the following format:

```
; This is a comment line
DB2INSTANCE=instancename
DB2DBDFT=database
; End of file
```

You can specify all the environment variables that you need to initialize the environment for `rah`.

Related concepts:

- "Specifying the rah and db2_all commands" on page 419

Determining problems with rah on Linux and UNIX platforms

Note: The information in this section applies to Linux and UNIX platforms only.
Here are suggestions on how to handle some problems that you might encounter when you are running `rah`:

1. `rah` hangs (or takes a very long time)

   This problem might be caused because:

   - `rah` has determined that it needs to buffer output, and you did not export `RAHCHECKBUF=no`. Therefore, before running your command, `rah` sends a command to all computers to check the existence of the buffer directory, and to create it if it does not exist.
   - One or more of the computers where you are sending your command is not responding. The `rsh` command will eventually time out but the time-out interval is quite long, usually about 60 seconds.
2. You have received messages such as:

- Login incorrect
- Permission denied

Either one of the computers does not have the ID running `rah` correctly defined in its .hosts file, or the ID running `rah` does not have one of the computers correctly defined in its .rhosts file. If the DB2RSHCMD registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.

Note: You might have a need to have greater security regarding the transmission of passwords in clear text between database partitions. This will depend on the remote shell program you are using. `rah` uses the remote shell program specified by the DB2RSHCMD registry variable. You can select between the two remote shell programs: ssh (for additional security), or rsh (or remsh for HP-UX). If this registry variable is not set, rsh (or remsh for HP-UX) is used.

3. When running commands in parallel using background remote shells, although the commands run and complete within the expected elapsed time at the computers, `rah` takes a long time to detect this and put up the shell prompt.

   The ID running `rah` does not have one of the computers correctly defined in its .rhosts file, or if the DB2RSHCMD registry variable has been configured to use ssh, then the ssh clients and servers on each computer might not be configured correctly.
4. Although `rah` runs fine when run from the shell command line, if you run `rah` remotely using rsh, for example,

   ```
   rsh somewher -l $USER db2_kill
   ```

   `rah` never completes.

   This is normal. `rah` starts background monitoring processes, which continue to run after it has exited. Those processes will normally persist until all processes associated with the command you ran have themselves terminated. In the case of `db2_kill`, this means termination of all database managers. You can terminate the monitoring processes by finding the process whose command is `rahwaitfor` and kill <process_id>. Do not specify a signal number. Instead, use the default (15).
5. The output from `rah` is not displayed correctly, or `rah` incorrectly reports that $RAHBUFNAME does not exist, when multiple commands of `rah` were issued under the same $RAHUSER.

   This is because multiple concurrent executions of `rah` are trying to use the same buffer file (for example, $RAHBUFDIR/$RAHBUFNAME) for buffering the outputs. To prevent this problem, use a different $RAHBUFNAME for each concurrent `rah` command, for example in the following ksh:

   ```
   export RAHBUFNAME=rahout
   rah ";$command_1" &
   export RAHBUFNAME=rah2out
   rah ";$command_2" &
   ```

   or use a method that makes the shell choose a unique name automatically such as:

   ```
   RAHBUFNAME=rahout.$$ db2_all "....."
   ```

   Whatever method you use, you must ensure you clean up the buffer files at some point if disk space is limited. `rah` does not erase a buffer file at the end of execution, although it will erase and then re-use an existing file the next time you specify the same buffer file.

6. You entered

   ```
   rah '"print from ()'
   ```

   and received the message:

   ```
   ksh: syntax error at line 1 : (' unexpected
   ```

   Prerequisites for the substitution of () and ## are:

   - Use `db2_all`, not `rah`.
   - Ensure a RAHOSTFILE is used either by exporting RAHOSTFILE or by defaulting to your /sqllib/db2nodes.cfg file. Without these prerequisites, `rah` will leave the () and ## as is. You receive an error because the command `print from ()` is not valid.

   For a performance tip when running commands in parallel, use `|` rather than `|`&, and use `||` rather than `||`& or ; unless you truly need the function provided by &. Specifying & requires more remote shell commands and therefore degrades performance.

Related reference:

- "Controlling the rah command" on page 426

Chapter 35. Load in a partitioned database environment - overview

In a multi-partition database, large amounts of data are located across many database partitions. Distribution keys are used to determine on which database partition each portion of the data resides. The data must be *distributed* before it can be loaded at the correct database partition. When loading tables in a multi-partition database, the load utility can:

- Distribute input data in parallel.
- Load data simultaneously on corresponding database partitions.
- Transfer data from one system to another system.

Loading data into a multi-partition database takes place in two phases: A setup phase, where database partition resources such as table locks are acquired, and a load phase where the data is loaded into the database partitions. You can use the ISOLATE_PART_ERRS option of the LOAD command to select how errors will be handled during either of these phases, and how errors on one or more of the database partitions will affect the load operation on the database partitions that are not experiencing errors.

When loading data into a a multi-partition database you can use one of the following modes:

- PARTITION_AND_LOAD. Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.
- PARTITION_ONLY. Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. Each file includes a partition header that specifies how the data was distributed across the database partitions, and that the file can be loaded into the database using the LOAD_ONLY mode.
- LOAD_ONLY. Data is assumed to be already distributed across the database partitions; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already distributed across the database partitions, but the data file does not contain a partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dumpfile if the *dumpfile* file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning will be written to the load message file for that database partition.
- ANALYZE. An optimal distribution map with even distribution across all database partitions is generated.

Concepts and Terminology

The following terminology will be used when discussing the behavior and operation of the load utility in a partitioned database environment with multiple database partitions:

- The *coordinator partition* is the database partition to which the user connects to perform the load operation. In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, it is assumed that the data file resides on this database partition unless the CLIENT option of the LOAD command is specified. Specifying the CLIENT option of the LOAD command indicates that the data to be loaded resides on a remotely connected client.

- In the PARTITION_AND_LOAD, PARTITION_ONLY, and ANALYZE modes, the *pre-partitioning agent* reads the user data and distributes it in round-robin fashion to the *partitioning agents* which will distribute the data. This process is always performed on the coordinator partition. A maximum of one partitioning agent is allowed per database partition for any load operation.
- In the PARTITION_AND_LOAD, LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes, *load agents* run on each output database partition and coordinate the loading of data to that database partition.
- *Load to file agents* run on each output database partition during a PARTITION_ONLY load operation. They receive data from partitioning agents and write it to a file on their database partition.
- The SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the user exit.

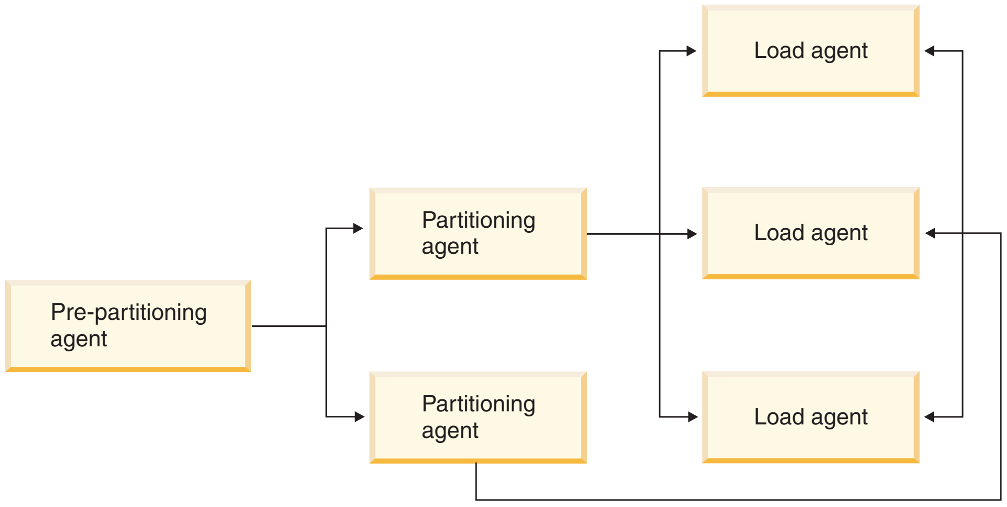


Figure 26. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, approximately half of the data is sent to each of two partitioning agents which distribute the data and send it to one of three database partitions. The load agent at each database partition loads the data.

Related concepts:

- "Data organization schemes" in *Administration Guide: Planning*
- "Load overview" in *Data Movement Utilities Guide and Reference*
- "Loading data in a partitioned database environment - hints and tips" in *Data Movement Utilities Guide and Reference*
- "Monitoring a load operation in a partitioned database environment using the LOAD QUERY command" in *Data Movement Utilities Guide and Reference*
- "Restarting or terminating a load operation in a partitioned database environment" in *Data Movement Utilities Guide and Reference*

Related tasks:

- "Loading data" in *Data Movement Utilities Guide and Reference*
- "Loading data in a partitioned database environment" in *Data Movement Utilities Guide and Reference*

Related reference:

- "Load configuration options for partitioned database environments" in *Data Movement Utilities Guide and Reference*

Chapter 36. DB2 UDB Commands for Administrators

ADD DBPARTITIONNUM

Adds a new database partition server to the partitioned database environment. This command also creates a database partition for all databases on the new database partition server. The user can specify the source database partition server for the definitions of any system temporary table spaces to be created with the new database partition, or specify that no system temporary table spaces are to be created. The command must be issued from the database partition server that is being added.

Scope:

This command only affects the machine on which it is executed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None

Command syntax:

```
►►──ADD DBPARTITIONNUM──┬─────────────────────────────────────────┬────────────►◄
                        ├─LIKE DBPARTITIONNUM──db-partition-number─┤
                        └─WITHOUT TABLESPACES──────────────────────┘
```

Command parameters:

LIKE DBPARTITIONNUM *db-partition-number*
: Specifies that the containers for the new system temporary table spaces are the same as the containers of the database at the database partition server specified by *db-partition-number*. The database partition server specified must already be defined in the `db2nodes.cfg` file.

  For system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the `MANAGED BY AUTOMATIC STORAGE` clause of the `CREATE TABLESPACE` statement or where no `MANAGED BY CLAUSE` was specified at all), the containers will not necessarily match those from the partition specified. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. This may or may not result in the same containers being used on these two partitions.

WITHOUT TABLESPACES
: Specifies that containers for the system temporary table spaces are not created for any of the database partitions. The ALTER TABLESPACE statement must be used to add system temporary table space containers to each database partition before the database can be used.

  If no option is specified, containers for the system temporary table spaces will be the same as the containers on the catalog partition for each database. The catalog partition can be a different database partition for each database in the partitioned database environment. This option is ignored for system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the `MANAGED BY AUTOMATIC STORAGE` clause of the `CREATE TABLESPACE` statement or where no `MANAGED BY CLAUSE` was specified at all). For these table spaces, there is no way to defer container creation. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database.

Usage notes:

Before adding a new database partition server, ensure that there is sufficient storage for the containers that must be created for all databases in the instance.

The add database partition server operation creates an empty database partition for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default values.

If an add database partition server operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added. Existing database partitions remain unaffected on all other database partition servers. If the clean-up phase fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition cannot contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition to a database partition group.

This command will fail if a create database or a drop database operation is in progress. The command can be reissued once the competing operation has completed.

This command will fail, if at any time in a database in the system a user table with an XML column has been created, successfully or not, or an XSR object has been registered, successfully or not.

To determine whether or not a database is enabled for automatic storage, `ADD DBPARTITIONNUM` has to communicate with the catalog partition for each of the databases in the instance. If automatic storage is enabled then the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, `ADD DBPARTITIONNUM` might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. The *start_stop_time* database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the command fails. If this situation occurs, increase the value of *start_stop_time*, and reissue the command.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODE can be substituted for DBPARTITIONNUM.

Related concepts:
- "Automatic storage databases" in *Administration Guide: Implementation*

Related reference:
- "START DATABASE MANAGER " on page 618

BACKUP DATABASE

Creates a backup copy of a database or a table space.

For information on the backup operations supported by DB2 database systems between different operating systems and hardware platforms, see ″Backup and restore operations between different operating systems and hardware platforms″ in the *Related concepts* section.

Scope:

This command only affects the database partition on which it is executed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

- *sysmaint*

Required connection:

Command syntax:

```
>>--BACKUP--+-DATABASE-+--database-alias--+-----------------------------------------+-------->
            '-DB-------'                  |                .-,---------------.      |
                                          |                V                 |      |
                                          '-TABLESPACE--(----tablespace-name-+--)--'

>--+----------------------------+------------------------------------------------------------>
   '-INCREMENTAL--+-----------+-'
                  '-DELTA-----'

>--+----------------------------------------------------------------------------------+------>
   +-USE--+-TSM--+--+------------------------------------+--+---------------------------------+-+
   |      '-XBSA-'  '-OPTIONS--+-"options-string"-+------'  '-OPEN--num-sessions--SESSIONS-'   |
   |                           '-@--file-name-----'                                            |
   |       .-,-------.                                                                         |
   |       V         |                                                                         |
   +-TO----+-dir-+---+-------------------------------------------------------------------------+
   |       '-dev-'                                                                             |
   '-LOAD--library-name--+------------------------------------+--+---------------------------------+-'
                         '-OPTIONS--+-"options-string"-+------'  '-OPEN--num-sessions--SESSIONS-'
                                    '-@--file-name-----'

>--+--------------------------------+--+---------------------------+--+------------------+--->
   '-WITH--num-buffers--BUFFERS-----'  '-BUFFER--buffer-size-------'  '-PARALLELISM--n---'

>--+-----------------------------------------------------------------------+----------------->
   '-COMPRESS--+--------------------------------+--+-----------------------+-'
               '-COMPRLIB--name--+-----------+--'  '-COMPROPTS--string-----'
                                 '-EXCLUDE---'

                                              .-EXCLUDE LOGS-.
>--+-----------------------------------+--+--------------+----------------------------------><
   '-UTIL_IMPACT_PRIORITY--+----------+-'  '-INCLUDE LOGS-'
                           '-priority-'
```

Command parameters:

DATABASE database-alias
: Specifies the alias of the database to back up.

TABLESPACE tablespace-name
: A list of names used to specify the table spaces to be backed up.

ONLINE

INCREMENTAL
: Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DELTA
: Specifies a non-cumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

USE TSM
: Specifies that the backup is to use Tivoli Storage Manager output.

USE XBSA
: Specifies that the XBSA interface is to be used. Backup Services APIs

(XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

OPTIONS

"options-string"
: Specifies options to be used for the backup operation.The string will be passed to the vendor support library, for example TSM, exactly as it was entered, without the quotes. Specifying this option overrides the value specified by the VENDOROPT database configuration parameter.

@file-name
: Specifies that the options to be used for the backup operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library, for example TSM. The file must be a fully qualified file name.

OPEN num-sessions SESSIONS
The number of I/O sessions to be created between DB2 and TSM or another backup vendor product. This parameter has no effect when backing up to tape, disk, or other local device.

TO dir/dev
A list of directory or tape device names.The full path on which the directory resides must be specified. This target directory or device must exist on the database server. This parameter can be repeated to specify the target directories and devices that the backup image will span. If more than one target is specified (target1, target2, and target3, for example), target1 will be opened first. The media header and special files (including the configuration file, table space table, and history file) are placed in target1. All remaining targets are opened, and are then used in parallel during the backup operation. Because there is no general tape support on Windows operating systems, each type of tape device requires a unique device driver. To back up to the FAT file system on Windows operating systems, users must conform to the 8.3 naming restriction.

c Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted)

d Device terminate. Stop using *only* the device that generated the warning message (for example, when there are no more tapes)

t Terminate. Abort the backup operation.

If the tape system does not support the ability to uniquely reference a backup image, it is recommended that multiple backup copies of the same database not be kept on the same tape.

LOAD library-name
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path on which the user exit program resides.

WITH num-buffers BUFFERS
The number of buffers to be used. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. However, when creating a backup to multiple locations, a larger number of buffers can be used to improve performance.

BUFFER buffer-size

The size, in 4 KB pages, of the buffer used when building the backup image. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

If using tape with variable block size, reduce the buffer size to within the range that the tape device supports. Otherwise, the backup operation might succeed, but the resulting image might not be recoverable.

With most versions of Linux, using DB2's default buffer size for backup operations to a SCSI tape device results in error SQL2025N, reason code 75. To prevent the overflow of Linux internal SCSI buffers, use this formula:

```
bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

where *bufferpages* is the value you want to use with the BUFFER parameter, and `ST_MAX_BUFFERS` and `ST_BUFFER_BLOCKS` are defined in the Linux kernel under the drivers/scsi directory.

PARALLELISM n

Determines the number of table spaces which can be read in parallel by the backup utility. DB2 will automatically choose an optimal value for this parameter unless you explicitly enter a value.

UTIL_IMPACT_PRIORITY *priority*

Specifies that the backup will run in throttled mode, with the priority specified. Throttling allows you to regulate the performance impact of the backup operation. Priority can be any number between 1 and 100, with 1 representing the lowest priority, and 100 representing the highest priority. If the UTIL_IMPACT_PRIORITY keyword is specified with no priority, the backup will run with the default priority of 50. If UTIL_IMPACT_PRIORITY is not specified, the backup will run in unthrottled mode. An impact policy must be defined by setting the *util_impact_lim* configuration parameter for a backup to run in throttled mode.

COMPRESS

Indicates that the backup is to be compressed.

COMPRLIB *name*

Indicates the name of the library to be used to perform the compression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, the default DB2 compression library will be used. If the specified library cannot be loaded, the backup will fail.

EXCLUDE

Indicates that the compression library will not be stored in the backup image.

COMPROPTS *string*

Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte reversal or code page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of string with the contents of this file and will pass this new value to the initialization routine instead. The maximum length for *string* is 1024 bytes.

EXCLUDE LOGS
Specifies that the backup image should not include any log files. When performing an offline backup operation, logs are excluded whether or not this option is specified.

INCLUDE LOGS
Specifies that the backup image should include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup.

WITHOUT PROMPTING

Examples:

Usage notes:

The data in a backup cannot be protected by the database server. Make sure that backups are properly safeguarded, particularly if the backup contains LBAC-protected data.

When backing up to tape, use of a variable block size is currently not supported. If you must use this option, ensure that you have well tested procedures in place that enable you to recover successfully, using backup images that were created with a variable block size.

When using a variable block size, you must specify a backup buffer size that is less than or equal to the maximum limit for the tape devices that you are using. For optimal performance, the buffer size must be equal to the maximum block size limit of the device being used.

Related concepts:

- "Backup and restore operations between different operating systems and hardware platforms" in *Data Recovery and High Availability Guide and Reference*
- "Developing a backup and recovery strategy" in *Data Recovery and High Availability Guide and Reference*

Related tasks:

- "Using backup" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "BACKUP DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

BIND

Invokes the bind utility, which prepares SQL statements stored in the bind file generated by the precompiler, and creates a package that is stored in the database.

Scope:

This command can be issued from any database partition in `db2nodes.cfg`. It updates the database catalogs on the catalog database partition. Its effects are visible to all database partitions.

Authorization:

One of the following:
- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax:

For DB2 for Windows and UNIX

```
►►──BIND──filename──────────────────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────────────────────┬─────►
   └─ACTION──┬─ADD────────────────────────────────────────────────────┘
             │ ┌─REPLACE─┐                                          │
             └─┴─────────┴──┬──────────────────┬──┬──────────────────────┬─┘
                            └─RETAIN──┬─NO──┬──┘  └─REPLVER──version-id─┘
                                      └─YES─┘

►──┬───────────────────────┬──┬──────────────────────────┬──┬─────────────────────────────┬──►
   └─BLOCKING──┬─UNAMBIG─┬─┘  └─CLIPKG──cli-packages─┘  └─COLLECTION──schema-name─┘
               ├─ALL─────┤
               └─NO──────┘

►──┬───────────────────┬──┬───────────────────────────────────┬─────────────►
   └─DATETIME──┬─DEF─┬─┘  └─DEGREE──┬─1──────────────────────┬─┘
               ├─EUR─┤              ├─degree-of-parallelism─┤
               ├─ISO─┤              └─ANY───────────────────┘
               ├─JIS─┤
               ├─LOC─┤
               └─USA─┘
```

```
►──┬─────────────────────────────────┬──┬───────────────────────┬──┬─────────────────────────┬──────────►
   └─DYNAMICRULES──┬─RUN────────┬──┘  └─EXPLAIN──┬─NO────┬─┘  └─EXPLSNAP──┬─NO────┬─┘
                   ├─BIND───────┤                ├─ALL───┤                 ├─ALL───┤
                   ├─INVOKERUN──┤                ├─REOPT─┤                 ├─REOPT─┤
                   ├─INVOKEBIND─┤                └─YES───┘                 └─YES───┘
                   ├─DEFINERUN──┤
                   └─DEFINEBIND─┘

►──┬───────────────────────┬──┬──────────────────────────────┬──┬──────────────────────┬──────────────────►
   └─FEDERATED──┬─NO──┬─┘  │              ┌─,───────────┐ │  └─GENERIC──string─┘
                └─YES─┘     └─FUNCPATH──▼─schema-name─┴─┘

►──┬──────────────────────────────────┬──┬─────────────────────┬──┬─────────────────────┬────────────────►
   ├─GRANT──┬─authid─┬───────────────┤  └─INSERT──┬─DEF─┬─┘  └─ISOLATION──┬─CS─┬─┘
   │        └─PUBLIC─┘               │            └─BUF─┘                   ├─RR─┤
   ├─GRANT_GROUP──group-name─────────┤                                      ├─RS─┤
   └─GRANT_USER──user-name───────────┘                                      └─UR─┘

►──┬─────────────────────────┬──┬──────────────────────────────┬──┬─────────────────────────────┬────────►
   └─MESSAGES──message-file─┘  └─OWNER──authorization-id─┘  └─QUALIFIER──qualifier-name─┘

                                      ┌─REOPT NONE───┐
►──┬─────────────────────────────────┬─┼──────────────┼──┬───────────────────────────┬──────────────────►
   └─QUERYOPT──optimization-level─┘  ├─REOPT ONCE───┤  └─SQLERROR──┬─CHECK─────┬─┘
                                      └─REOPT ALWAYS─┘               ├─CONTINUE──┤
                                                                     └─NOPACKAGE─┘

►──┬───────────────────────┬─────────────────────────────────────────────────────────────────────────────►
   └─SQLWARN──┬─NO──┬─┘
              └─YES─┘

►──┬──────────────────────────────────────────────────────────────────────────────────────┬──────────────►◄
   │                  ┌─NO──┐                                                                │
   └─STATICREADONLY──┴─YES─┴──┬───────────────────────┬──┬──────────────────────────────┬─┘
                                 └─VALIDATE──┬─BIND─┬─┘  └─TRANSFORM GROUP──groupname─┘
                                             └─RUN──┘
```

For DB2 on servers other than Windows and UNIX

```
►►──BIND──filename───────────────────────────────────────────────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────────────────────────────────────┬──────────────────►
   └─ACTION──┬─ADD──────────────────────────────────────────────────────────────────┤
             │  ┌─REPLACE─┐                                                          │
             └──┴─────────┴──┬──────────────────────┬──┬──────────────────────────┬─┘
                               └─RETAIN──┬─NO──┬─┘  └─REPLVER──version-id─┘
                                         └─YES─┘

►──┬───────────────────────────┬──┬─────────────────────────┬──┬───────────────────────┬────────────────►
   │            ┌─UNAMBIG─┐    │  └─CCSIDG──double-ccsid─┘  └─CCSIDM──mixed-ccsid─┘
   └─BLOCKING──┼─ALL─────┼────┘
                └─NO──────┘

►──┬───────────────────────┬──┬────────────────────────────┬──┬───────────────────────┬─────────────────►
   └─CCSIDS──sbcs-ccsid─┘  └─CHARSUB──┬─DEFAULT─┬─┘  └─CLIPKG──cli-packages─┘
                                        ├─BIT─────┤
                                        ├─MIXED───┤
                                        └─SBCS────┘
```

BIND

```
►─┬──────────────────────┬─┬────────────────────────────┬──────────────────────────►
  └─CNULREQD──┬─NO──┬──┘   └─COLLECTION──schema-name─┘
              └─YES─┘

►─┬──────────────────────────┬─┬───────────────────────────┬─┬────────────────┬────►
  │ (1)                      │ └─DBPROTOCOL──┬─DRDA────┬─┘   └─DEC──┬─15─┬─┘
  └───────────DATETIME──┬─DEF─┤               └─PRIVATE─┘           └─31─┘
                        ├─EUR─┤
                        ├─ISO─┤
                        ├─JIS─┤
                        ├─LOC─┤
                        └─USA─┘

►─┬──────────────────────────┬─┬────────────────────────────────────────────────┬──►
  └─DECDEL──┬─COMMA──┬─┘     │ (2)                                            │
            └─PERIOD─┘       └───────────DEGREE──┬─1─────────────────────┬─┘
                                                 ├─degree-of-parallelism─┤
                                                 └─ANY───────────────────┘

►─┬──────────────────────────────────┬─┬────────────────────────────┬──────────────►
  │                ┌─RUN────────┐    │ └─ENCODING──┬─ASCII───┬─┘
  └─DYNAMICRULES──┼─BIND───────┼──┘               ├─EBCDIC──┤
                   ├─INVOKERUN──┤                   ├─UNICODE─┤
                   ├─INVOKEBIND─┤                   └─CCSID───┘
                   ├─DEFINERUN──┤
                   └─DEFINEBIND─┘

►─┬─────────────────────────┬─┬──────────────────────┬─┬───────────────────────┬──►
  │ (3)                     │ └─GENERIC──string─┘      └─GRANT──┬─authid─┬─┘
  └───────────EXPLAIN──┬─NO──┤                                    └─PUBLIC─┘
                       └─YES─┘

►─┬────────────────────────┬─┬──────────────────────┬─┬────────────────────────┬───►
  └─IMMEDWRITE──┬─NO──┬─┘    └─INSERT──┬─BUF─┬─┘      │              ┌─CS─┐    │
                ├─YES─┤                └─DEF─┘        └─ISOLATION──┼─NC─┼──┘
                └─PH1─┘                                              ├─RR─┤
                                                                     ├─RS─┤
                                                                     └─UR─┘

►─┬─────────────────────────┬─┬───────────────────────────┬─┬─────────────────────┬►
  └─KEEPDYNAMIC──┬─YES─┬─┘    └─MESSAGES──message-file─┘    └─OPTHINT──hint-id─┘
                 └─NO──┘

►─┬────────────────────────┬─┬────────────────────────────────┬─┬─────────────────────┬►
  └─OS400NAMING──┬─SYSTEM─┬┘  └─OWNER──authorization-id─┘        └─PATH──schema-name─┘
                 └─SQL────┘

                                                                  ┌─REOPT NONE───┐
►─┬───────────────────────────────┬─┬───────────────────────────┬─┼──────────────┼─►
  └─QUALIFIER──qualifier-name─┘     └─RELEASE──┬─COMMIT─────┬─┘   ├─REOPT ONCE───┤
                                               └─DEALLOCATE─┘     └─REOPT ALWAYS─┘

►─┬───────────────┬─┬───────────────────────┬─┬──────────────────────────┬────────►
  ├─REOPT VARS────┤ └─SORTSEQ──┬─JOBRUN─┬─┘   └─SQLERROR──┬─CHECK─────┬─┘
  └─NOREOPT VARS──┘            └─HEX────┘                 ├─CONTINUE──┤
                                                          └─NOPACKAGE─┘
```

```
►─┬───────────────────────┬─┬──────────────────────────┬─┬──────────────┬──────────────►◄
  └─VALIDATE──┬─BIND─┬────┘ └─STRDEL──┬─APOSTROPHE─┬──┘ └─TEXT──label──┘
              └─RUN──┘                └─QUOTE──────┘
```

Notes:

1 If the server does not support the DATETIME DEF option, it is mapped to DATETIME ISO.

2 The DEGREE option is only supported by DRDA Level 2 Application Servers.

3 DRDA defines the EXPLAIN option to have the value YES or NO. If the server does not support the EXPLAIN YES option, the value is mapped to EXPLAIN ALL.

Command parameters:

filename

Specifies the name of the bind file that was generated when the application program was precompiled, or a list file containing the names of several bind files. Bind files have the extension `.bnd`. The full path name can be specified.

If a list file is specified, the @ character must be the first character of the list file name. The list file can contain several lines of bind file names. Bind files listed on the same line must be separated by plus (+) characters, but a + cannot appear in front of the first file listed on each line, or after the last bind file listed. For example,

```
/u/smith/sqllib/bnd/@all.lst
```

is a list file that contains the following bind files:

```
mybind1.bnd+mybind.bnd2+mybind3.bnd+
     mybind4.bnd+mybind5.bnd+
     mybind6.bnd+
     mybind7.bnd
```

ACTION

Indicates whether the package can be added or replaced.

ADD Indicates that the named package does not exist, and that a new package is to be created. If the package already exists, execution stops, and a diagnostic error message is returned.

REPLACE

Indicates that the existing package is to be replaced by a new one with the same package name and creator. This is the default value for the ACTION option.

RETAIN

Indicates whether BIND and EXECUTE authorities are to be preserved when a package is replaced. If ownership of the package changes, the new owner grants the BIND and EXECUTE authority to the previous package owner.

NO Does not preserve BIND and EXECUTE authorities when a package is replaced. This value is not supported by DB2.

YES Preserves BIND and EXECUTE authorities when a package is replaced. This is the default value.

REPLVER version-id
Replaces a specific version of a package. The version identifier specifies which version of the package is to be replaced. If the specified version does not exist, an error is returned. If the REPLVER option of REPLACE is not specified, and a package already exists that matches the package name, creator, and version of the package being bound, that package will be replaced; if not, a new package will be added.

BLOCKING
Specifies the type of row blocking for cursors.

ALL Specifies to block for:

- Read-only cursors
- Cursors not specified as FOR UPDATE OF

Ambiguous cursors are treated as read-only.

NO Specifies not to block any cursors. Ambiguous cursors are treated as updatable.

UNAMBIG
Specifies to block for:

- Read-only cursors
- Cursors not specified as FOR UPDATE OF

Ambiguous cursors are treated as updatable.

CCSIDG double-ccsid
An integer specifying the coded character set identifier (CCSID) to be used for double byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDM mixed-ccsid
An integer specifying the coded character set identifier (CCSID) to be used for mixed byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDS sbcs-ccsid
An integer specifying the coded character set identifier (CCSID) to be used for single byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CHARSUB
Designates the default character sub-type that is to be used for column definitions in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2 for Windows and UNIX.

BIT Use the FOR BIT DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

DEFAULT
Use the target system defined default in all new character columns for which an explicit sub-type is not specified.

MIXED
Use the FOR MIXED DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

SBCS Use the FOR SBCS DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

CLIPKG cli-packages
An integer between 3 and 30 specifying the number of CLI large packages to be created when binding CLI bind files against a database.

CNULREQD
This option is related to the LANGLEVEL precompile option, which is not supported by DRDA. It is valid only if the bind file is created from a C or a C++ application. This DRDA bind option is not supported by DB2 for Windows and UNIX.

NO The application was coded on the basis of the LANGLEVEL SAA1 precompile option with respect to the null terminator in C string host variables.

YES The application was coded on the basis of the LANGLEVEL MIA precompile option with respect to the null terminator in C string host variables.

COLLECTION schema-name
Specifies a 30-character collection identifier for the package. If not specified, the authorization identifier for the user processing the package is used.

DATETIME
Specifies the date and time format to be used.

DEF Use a date and time format associated with the territory code of the database.

EUR Use the IBM standard for Europe date and time format.

ISO Use the date and time format of the International Standards Organization.

JIS Use the date and time format of the Japanese Industrial Standard.

LOC Use the date and time format in local form associated with the territory code of the database.

USA Use the IBM standard for U.S. date and time format.

DBPROTOCOL
Specifies what protocol to use when connecting to a remote site that is identified by a three-part name statement. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

DEC Specifies the maximum precision to be used in decimal arithmetic operations. This DRDA precompile/bind option is not supported by DB2 for Windows and UNIX. The DRDA server will use a system defined default value if this option is not specified.

15 15-digit precision is used in decimal arithmetic operations.

31 31-digit precision is used in decimal arithmetic operations.

DECDEL
Designates whether a period (.) or a comma (,) will be used as the decimal point indicator in decimal and floating point literals. This DRDA precompile/bind option is not supported by DB2 for Windows and UNIX. The DRDA server will use a system defined default value if this option is not specified.

COMMA
Use a comma (,) as the decimal point indicator.

PERIOD
Use a period (.) as the decimal point indicator.

DEGREE
Specifies the degree of parallelism for the execution of static SQL statements in an SMP system. This option does not affect CREATE INDEX parallelism.

1 The execution of the statement will not use parallelism.

degree-of-parallelism
Specifies the degree of parallelism with which the statement can be executed, a value between 2 and 32 767 (inclusive).

ANY Specifies that the execution of the statement can involve parallelism using a degree determined by the database manager.

DYNAMICRULES
Defines which rules apply to dynamic SQL at run time for the initial setting of the values used for authorization ID and for the implicit qualification of unqualified object references.

RUN Specifies that the authorization ID of the user executing the package is to be used for authorization checking of dynamic SQL statements. The authorization ID will also be used as the default package qualifier for implicit qualification of unqualified object references within dynamic SQL statements. This is the default value.

BIND Specifies that all of the rules that apply to static SQL for authorization and qualification are to be used at run time. That is, the authorization ID of the package owner is to be used for authorization checking of dynamic SQL statements, and the default package qualifier is to be used for implicit qualification of unqualified object references within dynamic SQL statements.

DEFINERUN
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

DEFINEBIND
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking

and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

INVOKERUN
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

INVOKEBIND
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

Because dynamic SQL statements will be using the authorization ID of the package owner in a package exhibiting bind behavior, the binder of the package should not have any authorities granted to them that the user of the package should not receive. Similarly, when defining a routine that will exhibit define behavior, the definer of the routine should not have any authorities granted to them that the user of the package should not receive since a dynamic statement will be using the authorization ID of the routine’s definer.

The following dynamically prepared SQL statements cannot be used within a package that was not bound with DYNAMICRULES RUN: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY, and SET EVENT MONITOR STATE.

ENCODING
Specifies the encoding for all host variables in static statements in the plan or package. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

EXPLAIN
Stores information in the Explain tables about the access plans chosen for each SQL statement in the package. DRDA does not support the ALL value for this option.

NO Explain information will not be captured.

YES Explain tables will be populated with information about the chosen access plan at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as

MODIFIES SQL DATA. If this is not done, incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. In addition, explain information is gathered for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE register is set to NO.

If the package is to be used for a routine, the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ALL Explain information for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE register is set to `NO`.

If the package is to be used for a routine, the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985). This value for EXPLAIN is not supported by DRDA.

EXPLSNAP
Stores Explain Snapshot information in the Explain tables. This DB2 precompile/bind option is not supported by DRDA.

NO An Explain Snapshot will not be captured.

YES An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA or incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain snapshot information for each reoptimizable incremental bind SQL statement is placed in the explain tables at run time. In addition, explain snapshot information is gathered for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT register is set to NO.

If the package is to be used for a routine, the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ALL An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain snapshot information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, explain snapshot information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT register is set to `NO`.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

FEDERATED

Specifies whether a static SQL statement in a package references a nickname or a federated view. If this option is not specified and a static SQL statement in the package references a nickname or a federated view, a warning is returned and the package is created. This option is not supported for DRDA.

NO A nickname or federated view is not referenced in the static SQL statements of the package. If a nickname or federated view is encountered in a static SQL statement during the prepare or bind phase of this package, an error is returned and the package is *not* created.

YES A nickname or federated view can be referenced in the static SQL statements of the package. If no nicknames or federated views are encountered in static SQL statements during the prepare or bind of the package, no errors or warnings are returned and the package is created.

FUNCPATH

Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is ″SYSIBM″,″SYSFUN″,USER where USER is the value of the USER special register.

schema-name

An SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time. The same schema cannot appear more than once in the function path. The number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 254 bytes. The schema SYSIBM does not need to be explicitly specified; it is implicitly assumed to be the first schema if it is not included in the function path.

GENERIC string

Supports the binding of new options that are defined in the target database, but are not supported by DRDA. Do not use this option to pass bind options that *are* defined in BIND or PRECOMPILE. This option can substantially improve dynamic SQL performance. The syntax is as follows:

```
generic "option1 value1 option2 value2 ..."
```

Each option and value must be separated by one or more blank spaces. For example, if the target DRDA database is DB2 Universal Database, Version 8, one could use:

```
generic "explsnap all queryopt 3 federated yes"
```

to bind each of the EXPLSNAP, QUERYOPT, and FEDERATED options.

The maximum length of the string is 1023 bytes.

GRANT

authid Grants EXECUTE and BIND privileges to a specified user name or group ID.

PUBLIC
Grants EXECUTE and BIND privileges to PUBLIC.

GRANT_GROUP group-name
Grants EXECUTE and BIND privileges to a specified group ID.

GRANT_USER user-name
Grants EXECUTE and BIND privileges to a specified user name.

If more than one of the GRANT, GRANT_GROUP, and GRANT_USER options are specified, only the last option specified is executed.

INSERT
Allows a program being precompiled or bound against a DB2 Enterprise Server Edition server to request that data inserts be buffered to increase performance.

BUF Specifies that inserts from an application should be buffered.

DEF Specifies that inserts from an application should not be buffered.

ISOLATION
Determines how far a program bound to this package can be isolated from the effect of other executing programs.

CS Specifies Cursor Stability as the isolation level.

NC No Commit. Specifies that commitment control is not to be used. This isolation level is not supported by DB2 for Windows and UNIX.

RR Specifies Repeatable Read as the isolation level.

RS Specifies Read Stability as the isolation level. Read Stability ensures that the execution of SQL statements in the package is isolated from other application processes for rows read and changed by the application.

UR Specifies Uncommitted Read as the isolation level.

IMMEDWRITE
Indicates whether immediate writes will be done for updates made to group buffer pool dependent pagesets or database partitions. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

KEEPDYNAMIC
Specifies whether dynamic SQL statements are to be kept after commit points. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

MESSAGES message-file
Specifies the destination for warning, error, and completion status messages. A message file is created whether the bind is successful or not. If a message file name is not specified, the messages are written to standard output. If the complete path to the file is not specified, the current directory is used. If the name of an existing file is specified, the contents of the file are overwritten.

OPTHINT
: Controls whether query optimization hints are used for static SQL. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

OS400NAMING
: Specifies which naming option is to be used when accessing DB2 UDB for iSeries data. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

  Because of the slashes used as separators, a DB2 utility can still report a syntax error at execution time on certain SQL statements which use the iSeries system naming convention, even though the utility might have been precompiled or bound with the OS400NAMING SYSTEM option. For example, the Command Line Processor will report a syntax error on an SQL CALL statement if the iSeries system naming convention is used, whether or not it has been precompiled or bound using the OS400NAMING SYSTEM option.

OWNER authorization-id
: Designates a 30-character authorization identifier for the package owner. The owner must have the privileges required to execute the SQL statements contained in the package. Only a user with SYSADM or DBADM authority can specify an authorization identifier other than the user ID. The default value is the primary authorization ID of the precompile/bind process. SYSIBM, SYSCAT, and SYSSTAT are not valid values for this option.

PATH Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is "SYSIBM","SYSFUN",USER where USER is the value of the USER special register.

  schema-name
  : An SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time.

QUALIFIER qualifier-name
: Provides a 30-character implicit qualifier for unqualified objects contained in the package. The default is the owner's authorization ID, whether or not **owner** is explicitly specified.

QUERYOPT optimization-level
: Indicates the desired level of optimization for all static SQL statements contained in the package. The default value is 5. The SET CURRENT QUERY OPTIMIZATION statement describes the complete range of optimization levels available. This DB2 precompile/bind option is not supported by DRDA.

RELEASE
: Indicates whether resources are released at each COMMIT point, or when the application terminates. This DRDA precompile/bind option is not supported by DB2 for Windows and UNIX.

  COMMIT
  : Release resources at each COMMIT point. Used for dynamic SQL statements.

  DEALLOCATE
  : Release resources only when the application terminates.

SORTSEQ
Specifies which sort sequence table to use on the iSeries system. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

SQLERROR
Indicates whether to create a package or a bind file if an error is encountered.

CHECK
Specifies that the target system performs all syntax and semantic checks on the SQL statements being bound. A package will not be created as part of this process. If, while binding, an existing package with the same name and version is encountered, the existing package is neither dropped nor replaced even if **action replace** was specified.

CONTINUE
Creates a package, even if errors occur when binding SQL statements. Those statements that failed to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to execute them at run time generates an error (SQLCODE -525, SQLSTATE 51015).

NOPACKAGE
A package or a bind file is not created if an error is encountered.

REOPT
Specifies whether to have DB2 determine an access path at run time using values for host variables, parameter markers, and special registers. Valid values are:

NONE
The access path for a given SQL statement containing host variables, parameter markers, or special registers will not be optimized using real values. The default estimates for the these variables is used, and the plan is cached and will be used subsequently. This is the default value.

ONCE The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers, or special registers when the query is first executed. This plan is cached and used subsequently.

ALWAYS
The access path for a given SQL statement will always be compiled and reoptimized using the values of the host variables, parameter markers, or special registers that are known each time the query is executed.

REOPT / NOREOPT VARS
These options have been replaced by REOPT ALWAYS and REOPT NONE; however, they are still supported for back-level compatibility. Specifies whether to have DB2 determine an access path at run time using values for host variables, parameter markers, and special registers. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

SQLWARN
Indicates whether warnings will be returned from the compilation of

dynamic SQL statements (via PREPARE or EXECUTE IMMEDIATE), or from describe processing (via PREPARE...INTO or DESCRIBE).

NO Warnings will not be returned from the SQL compiler.

YES Warnings will be returned from the SQL compiler.

SQLCODE +238, +238 and +238 are exceptions. They are returned regardless of the **sqlwarn** option value.

STATICREADONLY
Determines whether static cursors will be treated as being READ ONLY. This DB2 precompile/bind option is not supported by DRDA.

NO All static cursors will take on the attributes as would normally be generated given the statement text and the setting of the LANGLEVEL precompile option. This is the default value.

YES Any static cursor that does not contain the FOR UPDATE or FOR READ ONLY clause will be considered READ ONLY.

STRDEL
Designates whether an apostrophe (′) or double quotation marks (") will be used as the string delimiter within SQL statements. This DRDA precompile/bind option is not supported by DB2 for Windows and UNIX. The DRDA server will use a system defined default value if this option is not specified.

APOSTROPHE
Use an apostrophe (′) as the string delimiter.

QUOTE
Use double quotation marks (") as the string delimiter.

TEXT label
The description of a package. Maximum length is 255 characters. The default value is blanks. This DRDA precompile/bind option is not supported by DB2 for Windows and UNIX.

TRANSFORM GROUP
Specifies the transform group name to be used by static SQL statements for exchanging user-defined structured type values with host programs. This transform group is not used for dynamic SQL statements or for the exchange of parameters and results with external functions or methods. This option is not supported by DRDA.

groupname
An SQL identifier of up to 18 characters in length. A group name cannot include a qualifier prefix and cannot begin with the prefix SYS since this is reserved for database use. In a static SQL statement that interacts with host variables, the name of the transform group to be used for exchanging values of a structured type is as follows:

- The group name in the TRANSFORM GROUP bind option, if any
- The group name in the TRANSFORM GROUP prep option as specified at the original precompilation time, if any
- The DB2_PROGRAM group, if a transform exists for the given type whose group name is DB2_PROGRAM

- No transform group is used if none of the above conditions exist.

The following errors are possible during the bind of a static SQL statement:

- SQLCODE yyyyy, SQLSTATE xxxxx: A transform is needed, but no static transform group has been selected.
- SQLCODE yyyyy, SQLSTATE xxxxx: The selected transform group does not include a necessary transform (TO SQL for input variables, FROM SQL for output variables) for the data type that needs to be exchanged.
- SQLCODE yyyyy, SQLSTATE xxxxx: The result type of the FROM SQL transform is not compatible with the type of the output variable, or the parameter type of the TO SQL transform is not compatible with the type of the input variable.

In these error messages, *yyyyy* is replaced by the SQL error code, and *xxxxx* by the SQL state code.

VALIDATE
Determines when the database manager checks for authorization errors and object not found errors. The package owner authorization ID is used for validity checking.

BIND Validation is performed at precompile/bind time. If all objects do not exist, or all authority is not held, error messages are produced. If **sqlerror continue** is specified, a package/bind file is produced despite the error message, but the statements in error are not executable.

RUN Validation is attempted at bind time. If all objects exist, and all authority is held, no further checking is performed at execution time.

If all objects do not exist, or all authority is not held at precompile/bind time, warning messages are produced, and the package is successfully bound, regardless of the **sqlerror continue** option setting. However, authority checking and existence checking for SQL statements that failed these checks during the precompile/bind process can be redone at execution time.

Examples:

The following example binds `myapp.bnd` (the bind file generated when the `myapp.sqc` program was precompiled) to the database to which a connection has been established:

```
db2 bind myapp.bnd
```

Any messages resulting from the bind process are sent to standard output.

Usage notes:

Binding a package using the REOPT option with the ONCE or ALWAYS value specified might change the static and dynamic statement compilation and performance.

Binding can be done as part of the precompile process for an application program source file, or as a separate step at a later time. Use BIND when binding is performed as a separate process.

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called `myapp.sql` generates a default bind file called `myapp.bnd` and a default package name of `MYAPP`. However, the bind file name and the package name can be overridden at precompile time by using the **bindfile** and the **package** options.

Binding a package with a schema name that does not already exist results in the implicit creation of that schema. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

BIND executes under the transaction that was started. After performing the bind, BIND issues a COMMIT or a ROLLBACK to terminate the current transaction and start another one.

Binding stops if a fatal error or more than 100 errors occur. If a fatal error occurs, the utility stops binding, attempts to close all files, and discards the package.

When a package exhibits bind behavior, the following will be true:

1. The implicit or explicit value of the BIND option OWNER will be used for authorization checking of dynamic SQL statements.
2. The implicit or explicit value of the BIND option QUALIFIER will be used as the implicit qualifier for qualification of unqualified objects within dynamic SQL statements.
3. The value of the special register CURRENT SCHEMA has no effect on qualification.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior as specified by the DYNAMICRULES option for that specific package and the environment they are used in.

Parameters displayed in the SQL0020W message are correctly noted as errors, and will be ignored as indicated by the message.

If an SQL statement is found to be in error and the BIND option SQLERROR CONTINUE was specified, the statement will be marked as invalid. In order to change the state of the SQL statement, another BIND must be issued . Implicit and explicit rebind will not change the state of an invalid statement. In a package bound with VALIDATE RUN, a statement can change from static to incremental bind or incremental bind to static across implicit and explicit rebinds depending on whether or not object existence or authority problems exist during the rebind.

Related concepts:

- "Isolation levels" in *SQL Reference, Volume 1*
- "Authorization considerations for dynamic SQL" on page 1307
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 1308
- "Performance improvements when using REOPT option of the BIND command" in *Developing Embedded SQL Applications*

Related reference:
- "SET CURRENT QUERY OPTIMIZATION statement" in *SQL Reference, Volume 2*
- "Datetime values" in *SQL Reference, Volume 1*
- "Special registers" on page 1523
- "DB2 CLI bind files and package names" in *Call Level Interface Guide and Reference, Volume 1*
- "PRECOMPILE " on page 635
- "sqlabndx - Bind application program to create a package" on page 779

CATALOG DATABASE

Stores database location information in the system database directory. The database can be located either on the local workstation or on a remote node.

Scope:

In a partitioned database environment, when cataloging a local database into the system database directory, this command must be issued from a database partition on the server where the database resides.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

None. Directory operations affect the local directory only.

Command syntax:

```
►►──CATALOG──┬─DATABASE─┬──database-name──┬───────────┬──┬─────────────────────┬──────►
             └─DB───────┘                  └─AS──alias─┘  ├─ON──┬─path──┬───────┤
                                                          │     └─drive─┘       │
                                                          └─AT NODE──nodename───┘

►──┬──────────────────────────────────────────────────────────┬────────────────────►
   └─AUTHENTICATION──┬─SERVER───────────────────────────────┬─┘
                     ├─CLIENT───────────────────────────────┤
                     ├─SERVER_ENCRYPT───────────────────────┤
                     ├─KERBEROS TARGET PRINCIPAL──principalname─┤
                     ├─DATA_ENCRYPT─────────────────────────┤
                     └─GSSPLUGIN────────────────────────────┘

►──┬───────────────────────────┬────────────────────────────────────────────────►◄
   └─WITH──"comment-string"────┘
```

Command parameters:

DATABASE *database-name*
: Specifies the name of the database to catalog.

AS *alias*
Specifies an alias as an alternate name for the database being cataloged. If an alias is not specified, the database manager uses *database-name* as the alias.

ON *path/drive*
Specifies the path on which the database being cataloged resides. On Windows operating systems, may instead specify the letter of the drive on which the database being cataloged resides (if it was created on a drive, not on a specific path).

AT NODE *nodename*
Specifies the name of the node where the database being cataloged resides. This name should match the name of an entry in the node directory. If the node name specified does not exist in the node directory, a warning is returned, but the database is cataloged in the system database directory. The node name should be cataloged in the node directory if a connection to the cataloged database is desired.

AUTHENTICATION

The authentication value is stored for remote databases (it appears in the output from the `LIST DATABASE DIRECTORY` command) but it is not stored for local databases.

Specifying an authentication type can result in a performance benefit.

SERVER
Specifies that authentication takes place on the node containing the target database.

CLIENT
Specifies that authentication takes place on the node where the application is invoked.

SERVER_ENCRYPT
Specifies that authentication takes place on the node containing the target database, and that passwords are encrypted at the source. Passwords are decrypted at the target, as specified by the authentication type cataloged at the source.

KERBEROS
Specifies that authentication takes place using Kerberos Security Mechanism. When authentication is Kerberos, only SECURITY=NONE is supported.

TARGET PRINCIPAL *principalname*
Fully qualified Kerberos principal name for the target server; that is, the fully qualified Kerberos principal of the DB2 instance owner in the form of `name/instance@REALM`. For Windows 2000, Windows XP, and Windows Server 2003, this is the logon account of the DB2 server service in the form of *`userid@DOMAIN`*, *`userid`*`@xxx.xxx.xxx.com` or *`domain\userid`*.

DATA_ENCRYPT
Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption.

GSSPLUGIN
Specifies that authentication takes place using an external GSS

API-based plug-in security mechanism. When authentication is GSSPLUGIN, only `SECURITY=NONE` is supported.

WITH *"comment-string"*
: Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

Examples:

```
db2 catalog database sample on /databases/sample
   with "Sample Database"
```

Usage notes:

Use CATALOG DATABASE to catalog databases located on local or remote nodes, recatalog databases that were uncataloged previously, or maintain multiple aliases for one database (regardless of database location).

DB2 automatically catalogs databases when they are created. It catalogs an entry for the database in the local database directory and another entry in the system database directory. If the database is created from a remote client (or a client which is executing from a different instance on the same machine), an entry is also made in the system database directory at the client instance.

If neither path nor node name is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter *dftdbpath*.

Databases on the same node as the database manager instance are cataloged as *indirect* entries. Databases on other nodes are cataloged as *remote* entries.

CATALOG DATABASE automatically creates a system database directory if one does not exist. The system database directory is stored on the path that contains the database manager instance that is being used, and is maintained outside of the database.

List the contents of the system database directory using the LIST DATABASE DIRECTORY command. To list the contents of the local database directory use the LIST DATABASE DIRECTORY ON /PATH, where PATH is where the database was created.

If directory caching is enabled, database and node directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications might not be effective until the application has restarted.

To refresh the CLP's directory cache, use the TERMINATE command. To refresh DB2's shared cache, stop (**db2stop**) and then restart (**db2start**) the database manager. To refresh the directory cache for another application, stop and then restart that application.

Related tasks:

- "Cataloging a database" in *Administration Guide: Implementation*

Related reference:

- "sqlecadb - Catalog a database in the system database directory" on page 794
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "LIST DATABASE DIRECTORY command" in *Command Reference*
- "TERMINATE command" in *Command Reference*
- "UNCATALOG DATABASE command" in *Command Reference*

CREATE DATABASE

The `CREATE DATABASE` command initializes a new database with an optional user-defined collating sequence, creates the three initial table spaces, creates the system tables, and allocates the recovery log file. When you initialize a new database, the `AUTOCONFIGURE` command is issued by default.

When the `CREATE DATABASE` command is issued, the Configuration Advisor also runs automatically. This means that the database configuration parameters are automatically tuned for you according to your system resources. In addition, Automated Runstats is enabled. To disable the Configuration Advisor from running at database creation, please refer to the *db2_enable_autoconfig_default* registry variable. To disable Automated Runstats, please refer to *auto_runstats* database configuration parameter.

Adaptive Self Tuning Memory is also enabled by default for single partition databases. To disable Adaptive Self Tuning Memory by default, refer to the *self_tuning_mem* database configuration parameter (see the *Related reference* section). For multi-partition databases, Adaptive Self Tuning Memory is disabled by default.

In order to make use of XML features, the codeset must be set to UTF-8 through the `USING CODESET` option of this command. In a future release of the DB2 database system, the default code set will be changed to UTF-8 when creating a database. If a particular code set and territory is needed for a database, the desired code set and territory should be specified in the `CREATE DATABASE` command

This command is not valid on a client.

Scope:

In a partitioned database environment, this command affects all database partitions that are listed in the `db2nodes.cfg` file.

The database partition from which this command is issued becomes the catalog database partition for the new database.

Authorization:

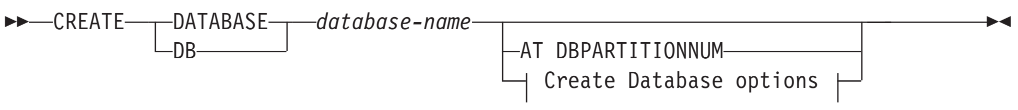
You must have one of the following:

- *sysadm*
- *sysctrl*

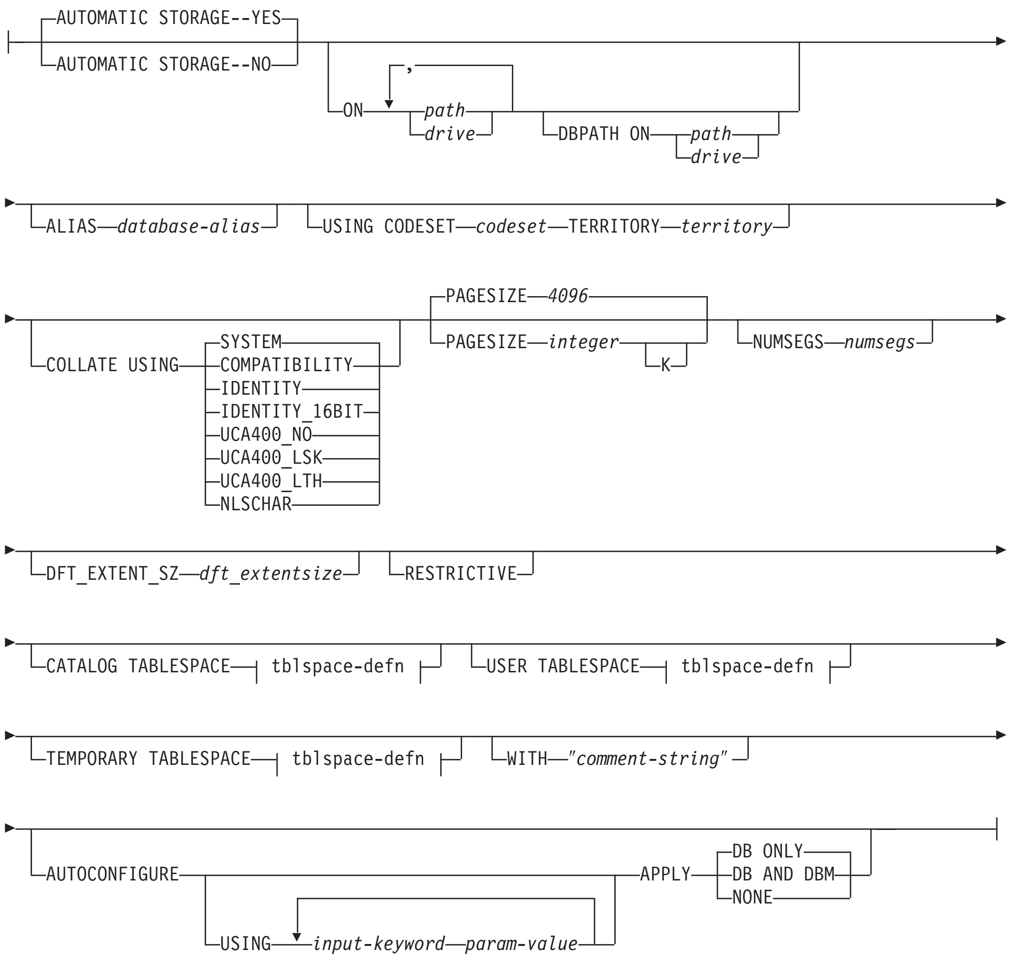
Required connection:

Instance. To create a database at another (remote) node, you must first attach to that node. A database connection is temporarily established by this command during processing.

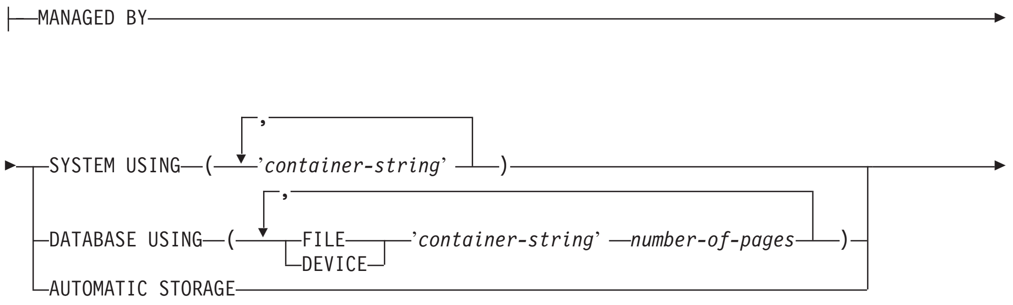
Command syntax:



Create Database options:



tblspace-defn:



```
►──┬──────────────────────────────┬──┬─────────────────────────────────┬──────────────────►
   └─EXTENTSIZE──number-of-pages──┘  └─PREFETCHSIZE──number-of-pages───┘

►──┬─────────────────────────────────────┬──┬──────────────────────────────────────────┬──►
   └─OVERHEAD──number-of-milliseconds────┘  └─TRANSFERRATE──number-of-milliseconds─────┘

►──┬──────────────────────┬──┬────────────────────────────────┬───────────────────────────►
   └─AUTORESIZE──┬─NO──┬──┘  └─INITIALSIZE──integer──┬─K─┬───┘
                 └─YES─┘                             ├─M─┤
                                                     └─G─┘

►──┬─────────────────────────────────────────┬──┬────────────────────────────────────┬────┤
   └─INCREASESIZE──integer──┬─PERCENT─┬──────┘  └─MAXSIZE──┬─NONE──────────────────┬─┘
                            ├─K───────┤                    └─integer──┬─K─┬────────┘
                            ├─M───────┤                               ├─M─┤
                            └─G───────┘                               └─G─┘
```

Notes:

1. The combination of the code set and territory values must be valid.
2. Not all collating sequences are valid with every code set and territory combination.
3. The table space definitions specified on CREATE DATABASE apply to all database partitions on which the database is being created. They cannot be specified separately for each database partition. If the table space definitions are to be created differently on particular database partitions, the CREATE TABLESPACE statement must be used.

   When defining containers for table spaces, $N can be used. $N will be replaced by the database partition number when the container is actually created. This is required if the user wants to specify containers in a multiple logical partition database.
4. The AUTOCONFIGURE option requires *sysadm* authority.

Command parameters:

DATABASE database-name
: A name to be assigned to the new database. This must be a unique name that differentiates the database from any other database in either the local database directory or the system database directory. The name must conform to naming conventions for databases. Specifically, the name must not contain any space characters.

AT DBPARTITIONNUM
: Specifies that the database is to be created only on the database partition that issues the command. You do not specify this option when you create a new database. You can use it to recreate a database partition that you dropped because it was damaged. After you use the CREATE DATABASE command with the AT DBPARITIONNUM option, the database at this database partition is in the restore-pending state. You must immediately restore the database on this node. This parameter is not intended for general use. For example, it should be used with RESTORE DATABASE command if the database partition at a node was damaged and must be re-created. Improper use of this parameter can cause inconsistencies in the system, so it should only be used with caution.

If this parameter is used to recreate a database partition that was dropped (because it was damaged), the database at this database partition will be in the restore-pending state. After recreating the database partition, the database must immediately be restored on this database partition.

AUTOMATIC STORAGE NO|YES
Specifies that automatic storage is being explicitly disabled or enabled for the database. The default value is `YES`. If the `AUTOMATIC STORAGE` clause is not specified, automatic storage is implicitly enabled by default.

NO Automatic storage is not being enabled for the database.

YES Automatic storage is being enabled for the database.

ON path or drive
The meaning of this option depends on the value of the AUTOMATIC STORAGE option.

- If AUTOMATIC STORAGE NO is specified, automatic storage is disabled for the database. In this case, only one path can be included as part of the ON option, and it specifies the path on which to create the database. If a path is not specified, the database is created on the default database path that is specified in the database manager configuration file (*dftdbpath* parameter). This behavior matches that of DB2 UDB Version 8.2 and earlier.
- Otherwise, automatic storage is enabled for the database by default. In this case, multiple paths may be listed here, each separated by a comma. These are referred to as storage paths and are used to hold table space containers for automatic storage table spaces. For multi-partition databases the same storage paths will be used on all partitions.

  With multiple paths, the DBPATH ON option specifies which of the multiple paths on which to create the database. If the DBPATH ON option is not specified, the database is created on the first path listed. If no paths are specified, the database is created on the default database path that is specified in the database manager configuration file (*dftdbpath* parameter). This will also be used as the location for the single storage path associated with the database.

The maximum length of a path is 175 characters.

For MPP systems, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the *dftdbpath* database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the `$HOME` directory of the instance owner). The path specified for this command in an MPP system cannot be a relative path. Also, all paths specified as part of the ON option must exist on all database partitions.

A given database path or storage path must exist and be accessible on each database partition.

DBPATH ON path or drive
If automatic storage is enabled, the DBPATH ON option specifies the path on which to create the database. If automatic storage is enabled and the DBPATH ON option is not specified, the database is created on the first path listed with the ON option.

The maximum length of a database path is 215 characters and the maximum length of a storage path is 175 characters.

ALIAS database-alias
An alias for the database in the system database directory. If no alias is provided, the specified database name is used.

USING CODESET codeset
Specifies the code set to be used for data entered into this database. After you create the database, you cannot change the specified code set. To make use of XML features, the codeset must be set to UTF-8.

TERRITORY territory
Specifies the territory to be used for data entered into this database. After you create the database, you cannot change the specified territory.

COLLATE USING
Identifies the type of collating sequence to be used for the database. Once the database has been created, the collating sequence cannot be changed.

COMPATIBILITY
The DB2 Version 2 collating sequence. Some collation tables have been enhanced. This option specifies that the previous version of these tables is to be used.

IDENTITY
Identity collating sequence, in which strings are compared byte for byte. This is the default for Unicode databases.

IDENTITY_16BIT
CESU-8 (Compatibility Encoding Scheme for UTF-16: 8-Bit) collation sequence as specified by the Unicode Technical Report #26, which is available at the Unicode Corsortium Web site (www.unicode.org). This option can only be specified when creating a Unicode database.

UCA400_NO
The UCA (Unicode Collation Algorithm) collation sequence that is based on the Unicode Standard version 4.00 with normalization implicitly set to on. Details of the UCA can be found in the Unicode Technical Standard #10, which is available at the Unicode Consortium Web site (www.unicode.org). This is the default collation when codeset UTF-8 is specified and can only be used when creating a Unicode database.

UCA400_LSK
The UCA (Unicode Collation Algorithm) collation sequence based on the Unicode Standard version 4.00 but will sort Slovakian characters in the appropriate order. Details of the UCA can be found in the Unicode Technical Standard #10, which is available at the Unicode Consortium Web site (www.unicode.org). This option can only be used when creating a Unicode database.

UCA400_LTH
The UCA (Unicode Collation Algorithm) collation sequence that is based on the Unicode Standard version 4.00 but will sort all Thai characters according to the Royal Thai Dictionary order. Details of the UCA can be found in the Unicode Technical Standard #10 available at the Unicode Consortium Web site (www.unicode.org). This option can only be used when creating a Unicode database. This collator might order Thai data differently from the NLSCHAR collator option.

NLSCHAR
System-defined collating sequence using the unique collation rules for the specific code set/territory.

This option can only be used with the Thai code page (CP874). If this option is specified in non-Thai environments, the command will fail and return the error SQL1083N with Reason Code 4.

SYSTEM
For non-Unicode databases, this is the default option, with the collating sequence based on the database territory. For Unicode databases, this option is equivalent to the `IDENTITY` option.

PAGESIZE integer
Specifies the page size of the default buffer pool along with the initial table spaces (SYSCATSPACE, TEMPSPACE1, USERSPACE1) when the database is created. This also represents the default page size for all future CREATE BUFFERPOOL and CREATE TABLESPACE statements. The valid values for integer without the suffix K are 4 096, 8 192, 16 384, or 32 768. The valid values for integer with the suffix K are 4, 8, 16, or 32. At least one space is required between the integer and the suffix K. The default is a page size of 4 096 bytes (4 K).

NUMSEGS numsegs
Specifies the number of directories (tablespace containers) that will be created and used to store the database table files for any default SMS table spaces. This parameter does not affect automatic storage table spaces, DMS table spaces, any SMS table spaces with explicit creation characteristics (created when the database is created), or any SMS table spaces explicitly created after the database is created.

DFT_EXTENT_SZ dft_extentsize
Specifies the default extent size of table spaces in the database.

RESTRICTIVE
If the RESTRICTIVE option is present it causes the RESTRICT_ACCESS database configuration parameter to be set to YES and no privileges are automatically granted to PUBLIC. If the RESTRICTIVE option is not present then the RESTRICT_ACCESS database configuration parameter is set to NO and all of the following privileges are automatically granted to PUBLIC.

- CREATETAB
- BINDADD
- CONNECT
- IMPLSCHEMA
- EXECUTE with GRANT on all procedures in schema SQLJ
- EXECUTE with GRANT on all functions and procedures in schema SYSPROC
- BIND on all packages created in the NULLID schema
- EXECUTE on all packages created in the NULLID schema
- CREATEIN on schema SQLJ
- CREATEIN on schema NULLID
- USE on table space USERSPACE1
- SELECT access to the SYSIBM catalog tables
- SELECT access to the SYSCAT catalog views

- SELECT access to the SYSSTAT catalog views
- UPDATE access to the SYSSTAT catalog views

CATALOG TABLESPACE tblspace-defn

Specifies the definition of the table space that will hold the catalog tables, SYSCATSPACE. If not specified and automatic storage is not enabled for the database, SYSCATSPACE is created as a System Managed Space (SMS) table space with *numsegs* number of directories as containers, and with an extent size of *dft_extentsize*. For example, the following containers would be created if *numsegs* were specified to be 5:

```
/u/smith/smith/NODE0000/SQL00001/SQLT0000.0
/u/smith/smith/NODE0000/SQL00001/SQLT0000.1
/u/smith/smith/NODE0000/SQL00001/SQLT0000.2
/u/smith/smith/NODE0000/SQL00001/SQLT0000.3
/u/smith/smith/NODE0000/SQL00001/SQLT0000.4
```

If not specified and automatic storage is enabled for the database, SYSCATSPACE is created as an automatic storage table space with its containers created on the defined storage paths. The extent size of this table space is 4. Appropriate values for AUTORESIZE, INITIALSIZE, INCREASESIZE, and MAXSIZE are set automatically.

See the CREATE TABLESPACE statement for more information on the table space definition fields.

In a partitioned database environment, the catalog table space is only created on the catalog database partition, the database partition on which the CREATE DATABASE command is issued.

USER TABLESPACE tblspace-defn

Specifies the definition of the initial user table space, USERSPACE1. If not specified and automatic storage is not enabled for the database, USERSPACE1 is created as an SMS table space with *numsegs* number of directories as containers and with an extent size of *dft_extentsize*. For example, the following containers would be created if *numsegs* were specified to be 5:

```
/u/smith/smith/NODE0000/SQL00001/SQLT0001.0
/u/smith/smith/NODE0000/SQL00001/SQLT0002.1
/u/smith/smith/NODE0000/SQL00001/SQLT0002.2
/u/smith/smith/NODE0000/SQL00001/SQLT0002.3
/u/smith/smith/NODE0000/SQL00001/SQLT0002.4
```

If not specified and automatic storage is enabled for the database, USERSPACE1 is created as an automatic storage table space with its containers created on the defined storage paths. The extent size of this table space will be *dft_extentsize*. Appropriate values for AUTORESIZE, INITIALSIZE, INCREASESIZE, and MAXSIZE are set automatically.

See the CREATE TABLESPACE statement for more information on the table space definition fields.

TEMPORARY TABLESPACE tblspace-defn

Specifies the definition of the initial system temporary table space, TEMPSPACE1. If not specified and automatic storage is not enabled for the database, TEMPSPACE1 is created as an SMS table space with *numsegs* number of directories as containers and with an extent size of *dft_extentsize*. For example, the following containers would be created if *numsegs* were specified to be 5:

```
/u/smith/smith/NODE0000/SQL00001/SQLT0002.0
/u/smith/smith/NODE0000/SQL00001/SQLT0001.1
/u/smith/smith/NODE0000/SQL00001/SQLT0001.2
/u/smith/smith/NODE0000/SQL00001/SQLT0001.3
/u/smith/smith/NODE0000/SQL00001/SQLT0001.4
```

If not specified and automatic storage is enabled for the database, TEMPSPACE1 is created as an automatic storage table space with its containers created on the defined storage paths. The extent size of this table space is *dft_extentsize*.

See the CREATE TABLESPACE statement for more information on the table space definition fields.

WITH *comment-string*
Describes the database entry in the database directory. Any comment that helps to describe the database can be entered. Maximum length is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by single or double quotation marks.

AUTOCONFIGURE
Based on user input, calculates the recommended settings for buffer pool size, database configuration, and database manager configuration and optionally applies them. The Configuration Advisor is run by default when the `CREATE DATABASE` command is issued. The `AUTOCONFIGURE` option is needed only if you want to tweaks the recommendations.

USING input-keyword param-value

Table 122. Valid input keywords and parameter values

Keyword	Valid values	Default value	Explanation
mem_percent	1–100	25	Percentage of memory to dedicate. If other applications (other than the operating system) are running on this server, set this to less than 100.
workload_type	simple, mixed, complex	mixed	Simple workloads tend to be I/O intensive and mostly transactions, whereas complex workloads tend to be CPU intensive and mostly queries.
num_stmts	1–1 000 000	25	Number of statements per unit of work
tpm	1–200 000	60	Transactions per minute
admin_priority	performance, recovery, both	both	Optimize for better performance (more transactions per minute) or better recovery time

Table 122. Valid input keywords and parameter values (continued)

Keyword	Valid values	Default value	Explanation
num_local_apps	0–5 000	0	Number of connected local applications
num_remote_apps	0–5 000	100	Number of connected remote applications
isolation	RR, RS, CS, UR	RR	Isolation level of applications connecting to this database (Repeatable Read, Read Stability, Cursor Stability, Uncommitted Read)
bp_resizeable	yes, no	yes	Are buffer pools resizeable?

APPLY

DB ONLY
Displays the recommended values for the database configuration and the buffer pool settings based on the current database manager configuration. Applies the recommended changes to the database configuration and the buffer pool settings.

DB AND DBM
Displays and applies the recommended changes to the database manager configuration, the database configuration, and the buffer pool settings.

NONE
Disables the Configuration Advisor (it is enabled by default).

- If the `AUTOCONFIGURE` keyword is specified with the `CREATE DATABASE` command, the `DB2_ENABLE_AUTOCONFIG_DEFAULT` variable value is not considered. Adaptive Self Tuning Memory and Auto Runstats will be enabled and the Configuration Advisor will tune the database configuration and database manager configuration parameters as indicated by the `APPLY DB` or `APPLY DBM` options.
- Specifying the `AUTOCONFIGURE` option with the `CREATE DATABASE` command on a database will recommend enablement of the Self Tuning Memory Manager. However, if you run the `AUTOCONFIGURE` command on a database in an instance where `SHEAPTHRES` is not zero, sort memory tuning (`SORTHEAP`) will not be enabled automatically. To enable sort memory tuning (`SORTHEAP`), you must set `SHEAPTHRES` equal to zero using the `UPDATE DATABASE MANAGER CONFIGURATION` command. Note that changing the value of `SHEAPTHRES` may affect the sort memory usage in your previously existing databases.

Usage notes:

The CREATE DATABASE command:

- Creates a database in the specified subdirectory. In a partitioned database environment, creates the database on all database partitions listed in `db2nodes.cfg`, and creates a `$DB2INSTANCE/NODE`*xxxx* directory under the specified subdirectory at each database partition. In a single partition database environment, creates a `$DB2INSTANCE/NODE0000` directory under the specified subdirectory.

- Creates the system catalog tables and recovery log.
- Catalogs the database in the following database directories:
  - Server's local database directory on the path indicated by *path* or, if the path is not specified, the default database path defined in the database manager system configuration file by the *dftdbpath* parameter. A local database directory resides on each file system that contains a database.
  - Server's system database directory for the attached instance. The resulting directory entry will contain the database name and a database alias.

    If the command was issued from a remote client, the client's system database directory is also updated with the database name and an alias.

  Creates a system or a local database directory if neither exists. If specified, the comment and code set values are placed in both directories.
- Stores the specified code set, territory, and collating sequence. A flag is set in the database configuration file if the collating sequence consists of unique weights, or if it is the identity sequence.
- Creates the schemas called SYSCAT, SYSFUN, SYSIBM, and SYSSTAT with SYSIBM as the owner. The database partition server on which this command is issued becomes the catalog database partition for the new database. Two database partition groups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP.
- Binds the previously defined database manager bind files to the database (these are listed in the utilities bind file list, `db2ubind.lst`). If one or more of these files do not bind successfully, CREATE DATABASE returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC.

  The utilities bind file list contains two bind files that cannot be bound against down-level servers:
  - `db2ugtpi.bnd` cannot be bound against DB2 Version 2 servers.
  - `db2dropv.bnd` cannot be bound against DB2 Parallel Edition Version 1 servers.

  If `db2ubind.lst` is bound against a down-level server, warnings pertaining to these two files are returned, and can be disregarded.
- Creates SYSCATSPACE, TEMPSPACE1, and USERSPACE1 table spaces. The SYSCATSPACE table space is only created on the catalog database partition.
- Grants the following:
  - EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema
  - EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema
  - DBADM authority, and CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA and LOAD privileges to the database creator
  - CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA privileges to PUBLIC
  - USE privilege on the USERSPACE1 table space to PUBLIC
  - SELECT privilege on each system catalog to PUBLIC
  - BIND and EXECUTE privilege to PUBLIC for each successfully bound utility.

– EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema.
– EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema.

Automatic storage is a collection of storage paths associated with a database on which table spaces can be created without having to explicitly specify container definitions (see the CREATE TABLESPACE statement for more information). Automatic storage is enabled by default, but can be explicitly disabled for a database when it is created. Automatic storage can be disbled at database creation time by specifying the AUTOMATIC STORAGE NO option.

It is important to note that automatic storage can only be enabled at database creation time, it cannot be enabled after the database has been created. Also, automatic storage cannot be disabled once a database has been defined to use it.

When free space is calculated for an automatic storage path for a given database partition, the database manager will check for the existence of the following directories or mount points within the storage path and will use the first one that is found. In doing this, file systems can be mounted at a point beneath the storage path and the database manager will recognize that the actual amount of free space available for table space containers may not be the same amount that is associated with the storage path directory itself.

1. `<storage path>/<instance name>/NODE####/<database name>`
2. `<storage path>/<instance name>/NODE####`
3. `<storage path>/<instance name>`
4. `<storage path>/<`

Where

- `<storage path>` is a storage path associated with the database.
- `<instance name>` is the instance under which the database resides.
- `NODE####` corresponds to the database partition number (for example NODE0000 or NODE0001).
- `<database name>` is the name of the database.

Consider the example where two logical database partitions exist on one physical machine and the database is being created with a single storage path: `/db2data`. Each database partition will use this storage path but the user may wish to isolate the data from each partition within its own file system. In this case, a separate file system can be created for each partition and be mounted at `/db2data/<instance>/NODE####`. When creating containers on the storage path and determining free space, the database manager will know not to retrieve free space information for `/db2data`, but instead retrieve it for the corresponding `/db2data/<instance>/NODE####` directory.

In general, the same storage paths must be used for each partition in a multi-partition database and they must all exist prior to executing the `CREATE DATABASE` command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You use the argument `" $N"` (`[blank]$N`) to indicate a database partition expression. A database partition expression can be used anywhere in the storage path, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path after the database partition expression is evaluated. If there is no space character in the storage path after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms:

Operators are evaluated from left to right. `%` represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N	" $N"	10
[blank]$N+[number]	" $N+100"	110
[blank]$N%[number]	" $N%5"	0
[blank]$N+[number]%[number]	" $N+1%5"	1
[blank]$N%[number]+[number]	" $N%4+2"	4
[a] % is modulus.		

With *dbadm* authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with *sysadm* or *dbadm* authority over the database revokes these privileges, the database creator nevertheless retains them.

In an MPP environment, the database manager creates a subdirectory, `$DB2INSTANCE/NODE`*xxxx*, under the specified or default path on all database partitions. The *xxxx* is the database partition number as defined in the `db2nodes.cfg` file (that is, database partition 0 becomes `NODE0000`). Subdirectories `SQL00001` through `SQL`*nnnnn* will reside on this path. This ensures that the database objects associated with different database partitions are stored in different directories (even if the subdirectory `$DB2INSTANCE` under the specified or default path is shared by all database partitions).

If LDAP (Lightweight Directory Access Protocol) support is enabled on the current machine, the database will be automatically registered in the LDAP directory. If a database object of the same name already exists in the LDAP directory, the database is still created on the local machine, but a warning message is returned, indicating that there is a naming conflict. In this case, the user can manually catalog an LDAP database entry by using the CATALOG LDAP DATABASE command.

CREATE DATABASE will fail if the application is already connected to a database.

When a database is created, a detailed deadlocks event monitor is created. As with any monitor, there is some overhead associated with this event monitor. You can drop the deadlocks event monitor by issuing the DROP EVENT MONITOR command.

Use CATALOG DATABASE to define different alias names for the new database.

Examples:

Here are several examples of the CREATE DATABASE command:

Example 1:

```
CREATE DATABASE TESTDB3
  AUTOMATIC STORAGE YES
```

Database TESTDB3 is created on the drive that is the value of database manager configuration parameter *dftdbpath*. Automatic storage is enabled with a single storage path that also has the value of *dftdbpath*.

Example 2:

```
CREATE DATABASE TESTDB7 ON C:,D:
```

Database TESTDB7 is created on drive C: (first drive in storage path list). Automatic storage is implicitly enabled and the storage paths are C: and D:.

Example 3:

```
CREATE DATABASE TESTDB15
 AUTOMATIC STORAGE YES
 ON C:,D: DBPATH ON E:
```

Database TESTDB15 is created on drive E: (explicitly listed as DBPATH). Automatic storage is explicitly enabled and the storage paths are C: and D:.

Example 4:

```
CREATE DATABASE TESTDB9 ON C:
 USING CODESET UTF-8 TERRITORY US
```

Database `TESTDB9` is created on drive `C:`. The codeset is set to `UTF-8`, enabling the use of native XML functionality on the database.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODE can be substituted for DBPARTITIONNUM.

Related concepts:
- "Isolation levels" in *SQL Reference, Volume 1*
- "Automatic storage databases" in *Administration Guide: Implementation*
- "Unicode implementation in DB2 Database for Linux, UNIX, and Windows" in *Administration Guide: Planning*

Related tasks:
- "Creating a database" on page 293
- "Securing the system catalog view" on page 166
- "Collating Thai characters" in *Administration Guide: Planning*

Related reference:
- "AUTOCONFIGURE command" in *Command Reference*
- "BIND" on page 441
- "CATALOG DATABASE " on page 458
- "CATALOG LDAP DATABASE command" in *Command Reference*
- "DROP DATABASE " on page 494
- "sqlecran API - Create a database on a database partition server" in *Administrative API Reference*
- "sqlecrea - Create database" on page 800
- "CREATE TABLESPACE " on page 1021
- "RESTORE DATABASE " on page 591

- “auto_maint - Automatic maintenance configuration parameter” in *Performance Guide*
- “Miscellaneous variables” in *Performance Guide*
- “self_tuning_mem- Self tuning memory configuration parameter” in *Performance Guide*

db2audit - Audit facility administrator tool

DB2 provides an audit facility to assist in the detection of unknown or unanticipated access to data. The DB2 audit facility generates and permits the maintenance of an audit trail for a series of predefined database events. The records generated from this facility are kept in an audit log file. The analysis of these records can reveal usage patterns which would identify system misuse. Once identified, actions can be taken to reduce or eliminate such system misuse. The audit facility acts at an instance level, recording all instance level activities and database level activities.

When working in a partitioned database environment, many of the auditable events occur at the database partition at which the user is connected (the coordinator partition) or at the catalog partition (if they are not the same database partition). The implication of this is that audit records can be generated by more than one database partition. Part of each audit record contains information on the coordinator partition and originating database partition identifiers.

The audit log (db2audit.log) and the audit configuration file (db2audit.cfg) are located in the instance’s `security` subdirectory. At the time you create an instance, read/write permissions are set on these files, where possible, by the operating system. By default, the permissions are read/write for the instance owner only. It is recommended that you do not change these permissions.

Authorized users of the audit facility can control the following actions within the audit facility, using `db2audit`:
- Start recording auditable events within the DB2 instance.
- Stop recording auditable events within the DB2 instance.
- Configure the behavior of the audit facility.
- Select the categories of the auditable events to be recorded.
- Request a description of the current audit configuration.
- Flush any pending audit records from the instance and write them to the audit log.
- Extract audit records by formatting and copying them from the audit log to a flat file or ASCII delimited files. Extraction is done for one of two reasons: In preparation for analysis of log records, or in preparation for pruning of log records.
- Prune audit records from the current audit log.

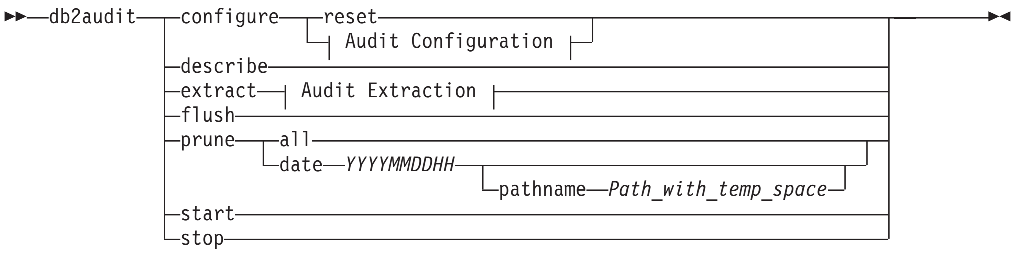
Authorization:

sysadm

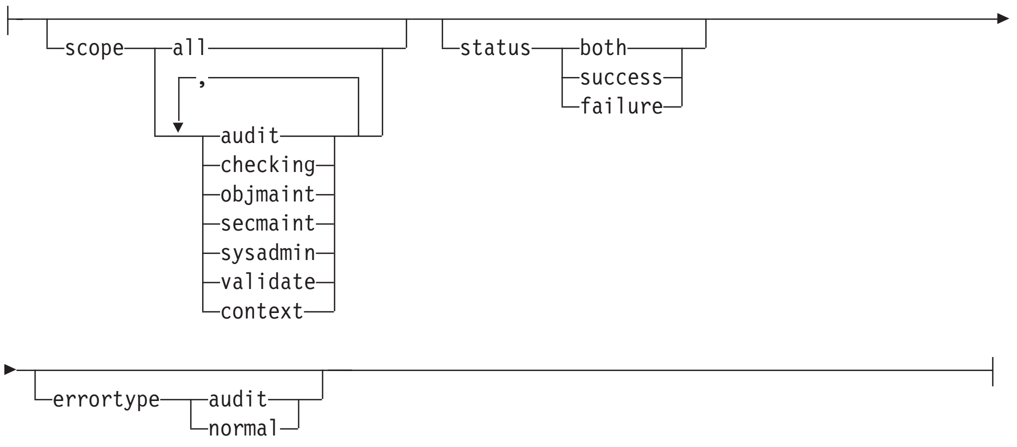
Required Connection:

None.

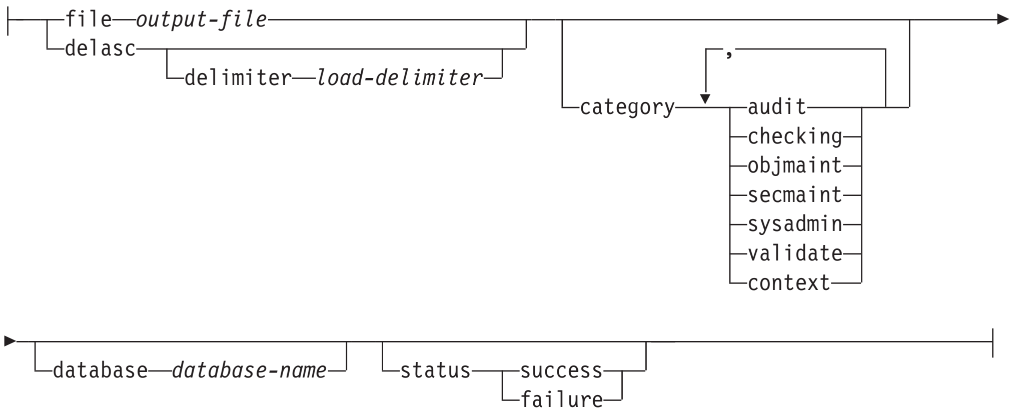
Command syntax:



Audit Configuration:



Audit Extraction:



Command parameters:

configure

This parameter allows the modification of the db2audit.cfg configuration file in the instance's security subdirectory. Updates to this file can occur even when the instance is shut down. Updates occurring when the instance is active dynamically affect the auditing being done by DB2 database across all database partitions. The configure action on the configuration file causes the creation of an audit record if the audit facility has been started and the *audit* category of auditable events is being audited.

The following are the possible actions on the configuration file:

- RESET. This action causes the configuration file to revert to the initial configuration (where SCOPE is all of the categories except CONTEXT, STATUS is FAILURE, ERRORTYPE is NORMAL, and the audit facility is OFF). This action will create a new audit configuration file if the original has been lost or damaged.
- SCOPE. This action specifies which category or categories of events are to be audited. This action also allows a particular focus for auditing and reduces the growth of the log. It is recommended that the number and type of events being logged be limited as much as possible, otherwise the audit log will grow rapidly. The default SCOPE is all categories except CONTEXT and may result in records being generated rapidly. In conjunction with the mode (synchronous or asynchronous), the selection of the categories may result in a significant performance reduction and significantly increased disk requirements.
- STATUS. This action specifies whether only successful or failing events, or both successful and failing events, should be logged. Context events occur before the status of an operation is known. Therefore, such events are logged regardless of the value associated with this parameter.
- ERRORTYPE. This action specifies whether audit errors are returned to the user or are ignored. The value for this parameter can be:
  - AUDIT. All errors including errors occurring within the audit facility are managed by DB2 database and all negative SQLCODEs are reported back to the caller.
  - NORMAL. Any errors generated by db2audit are ignored and only the SQLCODEs for the errors associated with the operation being performed are returned to the application.

describe
This parameter displays to standard output the current audit configuration information and status.

extract This parameter allows the movement of audit records from the audit log to an indicated destination. If no optional clauses are specified, all of the audit records are extracted and placed in a flat report file. If *output_file* already exists, an error message is returned.

The following are the possible options that can be used when extracting:

- FILE. The extracted audit records are placed in a file (*output_file*). If no file name is specified, records are written to the db2audit.out file in the security subdirectory of sqllib. If no directory is specified, *output_file* is written to the current working directory.
- DELASC. The extracted audit records are placed in a delimited ASCII format suitable for loading into DB2 database relational tables. The output is placed in separate files: one for each category. The filenames are:
  - audit.del
  - checking.del
  - objmaint.del
  - secmaint.del
  - sysadmin.del
  - validate.del
  - context.del

  These files are always written to the security subdirectory of sqllib.

  The DELASC choice also allows you to override the default audit character string delimiter (“0xff”) when extracting from the audit log.

You would use DELASC DELIMITER followed by the new delimiter that you wish to use in preparation for loading into a table that will hold the audit records. The new load delimiter can be either a single character (such as !) or a four-byte string representing a hexadecimal number (such as 0xff).

- CATEGORY. The audit records for the specified categories of audit events are to be extracted. If not specified, all categories are eligible for extraction.
- DATABASE. The audit records for a specified database are to be extracted. If not specified, all databases are eligible for extraction.
- STATUS. The audit records for the specified status are to be extracted. If not specified, all records are eligible for extraction.

flush This parameter forces any pending audit records to be written to the audit log. Also, the audit state is reset in the engine from “unable to log” to a state of “ready to log” if the audit facility is in an error state.

prune This parameter allows for the deletion of audit records from the audit log. If the audit facility is active and the “audit” category of events has been specified for auditing, then an audit record will be logged after the audit log is pruned.

The following are the possible options that can be used when pruning:

- ALL. All of the audit records in the audit log are to be deleted.
- DATE yyyymmddhh. The user can specify that all audit records that occurred on or before the date/time specified are to be deleted from the audit log. The user may optionally supply a

  ```
  pathname
  ```

  which the audit facility will use as a temporary space when pruning the audit log. This temporary space allows for the pruning of the audit log when the disk it resides on is full and does not have enough space to allow for a pruning operation.

start This parameter causes the audit facility to begin auditing events based on the contents of the db2audit.cfg file. In a partitioned DB2 database instance, auditing will begin on all database partitions when this clause is specified. If the “audit” category of events has been specified for auditing, then an audit record will be logged when the audit facility is started.

stop This parameter causes the audit facility to stop auditing events. In a partitioned DB2 database instance, auditing will be stopped on all database partitions when this clause is specified. If the “audit” category of events has been specified for auditing, then an audit record will be logged when the audit facility is stopped.

Usage Notes:

- The audit facility must be stopped and started explicitly. When starting, the audit facility uses existing audit configuration information. Since the audit facility is independent of the DB2 database server, it will remain active even if the instance is stopped. In fact, when the instance is stopped, an audit record may be generated in the audit log.
- Ensure that the audit facility has been turned on by issuing the db2audit start command before using the audit utilities.
- There are different categories of audit records that may be generated. In the description of the categories of events available for auditing (below), you should

notice that following the name of each category is a one-word keyword used to identify the category type. The categories of events available for auditing are:

- Audit (AUDIT). Generates records when audit settings are changed or when the audit log is accessed.
- Authorization Checking (CHECKING). Generates records during authorization checking of attempts to access or manipulate DB2 database objects or functions.
- Object Maintenance (OBJMAINT). Generates records when creating or dropping data objects.
- Security Maintenance (SECMAINT). Generates records when granting or revoking: object or database privileges, or DBADM authority. Records are also generated when the database manager security configuration parameters SYSADM_GROUP, SYSCTRL_GROUP, or SYSMAINT_GROUP are modified.
- System Administration (SYSADMIN). Generates records when operations requiring SYSADM, SYSMAINT, or SYSCTRL authority are performed.
- User Validation (VALIDATE). Generates records when authenticating users or retrieving system security information.
- Operation Context (CONTEXT). Generates records to show the operation context when a database operation is performed. This category allows for better interpretation of the audit log file. When used with the log's event correlator field, a group of events can be associated back to a single database operation. For example, a query statement for dynamic queries, a package identifier for static queries, or an indicator of the type of operation being performed, such as CONNECT, can provide needed context when analyzing audit results. The SQL or XQuery statement providing the operation context might be very long and is completely shown within the CONTEXT record. This can make the CONTEXT record very large.
- You can audit failures, successes, or both.

- Any operation on the database may generate several records. The actual number of records generated and moved to the audit log depends on the number of categories of events to be recorded as specified by the audit facility configuration. It also depends on whether successes, failures, or both, are audited. For this reason, it is important to be selective of the events to audit.

Related tasks:

- "Creating DB2 audit data files" on page 207

Related reference:

- "Audit facility usage" on page 200

db2gpmap - Get distribution map

If a database is already set up and database partition groups defined for it, `db2gpmap` gets the distribution map for the database table or the database partition group from the catalog partitioned database server.

Authorization:

Both of the following:

- Read access to the system catalog tables
- BIND and EXECUTE package privileges on db2gpmap.bnd

Required connection:

Before using `db2gpmap` the database manager must be started and db2gpmap.bnd must be bound to the database. If not already bound `db2gpmap` will attempt to bind the file.

Command syntax:

```
►►──db2gpmap──┬─────────────────────────┬──┬────────────────────────┬──────────────────►
              └─-d──┬───────────────┬───┘  └─-m──┬───────────────┬──┘
                    └─database-name─┘            └─map-file-name─┘

►──┬────────────────────────────────────────┬──┬─────────────────┬──┬────┬──────────►◄
   └─-g──┬─────────────────────────────────┬┘  └─-t──table-name──┘  └─-h─┘
         └─database-partition-group-name───┘
```

Command parameters:

-d Specifies the name of the database for which to generate a distribution map. If no database name is specified, the value of the DB2DBDFT environment variable is used. If DB2DBDFT is not set, the default is the SAMPLE database.

-m Specifies the fully qualified file name where the distribution map will be saved. The default is db2split.map.

-g Specifies the name of the database partition group for which to generate a distribution map. The default is IBMDEFAULTGROUP.

-t Specifies the table name.

-h Displays usage information.

Examples:

The following example extracts the distribution map for a table ZURBIE.SALES in database SAMPLE into a file called C:\pmaps\zurbie_sales.map:

```
db2gpmap -d SAMPLE -m C:\pmaps\zurbie_sales.map -t ZURBIE.SALES
```

Related concepts:

- "Distribution maps" on page 252

db2icrt - Create instance

Creates DB2 instances.

On Linux and UNIX-based systems, this utility is located in the `DB2DIR/instance` directory, where `DB2DIR` represents the installation location where the current version of the DB2 database system is installed. On Windows operating systems, this utility is located under the `DB2PATH\bin` directory where `DB2PATH` is the location where the DB2 copy is installed.

The `db2icrt` command creates DB2 instances related to the DB2 copy installation path from where `db2icrt` is issued.

Authorization:

Root access on Linux and UNIX-based systems or Local Administrator authority on Windows operating systems.

Command syntax:

For Linux and UNIX-based systems

```
►►──db2icrt──┬──────┬──┬────┬──┬──────────────┬──┬──────────────┬──────────►
             ├─ -h──┤  └─ -d┘  └─ -a──AuthType┘  └─ -p──PortName┘
             └─ -?──┘

►──┬──────────────┬──┬──────────────┬──InstName──────────────────────────►◄
   └─ -s──InstType┘  └─ -u──FencedID┘
```

For Windows operating systems

```
►►──db2icrt──InstName──┬──────────────┬──┬──────────────────────────┬──────►
                       └─ -s──InstType┘  └─ -u──UserName, Password──┘

►──┬──────────────────┬──┬──────────────┬──┬───────────────┬──┬──────┬─────►◄
   └─ -p──InstProfPath┘  └─ -h──HostName┘  └─ -r──PortRange┘  └─ -?──┘
```

Command parameters:
For Linux and UNIX-based systems

-h or -?
Displays the usage information.

-d
Turns debug mode on. Use this option only when instructed by DB2 Support.

-a *AuthType*
Specifies the authentication type (SERVER, CLIENT or SERVER_ENCRYPT) for the instance. The default is SERVER.

-p *PortName*
Specifies the port name or number used by the instance. This option does not apply to client instances.

-s *InstType*
Specifies the type of instance to create. Use the -s option only when you are creating an instance other than the default for your system. Valid values are:

client	Used to create an instance for a client. Use this value if you are using DB2 Connect Personal Edition.
ese	Used to create an instance for a database server with local and remote clients.
wse	Used to create an instance for DB2 Workgroup Server Edition, DB2 Express Edition and DB2 Connect Enterprise Edition.

-u *Fenced ID*
Specifies the name of the user ID under which fenced user-defined functions and fenced stored procedures will run. The -u option is requred if you are not creating a client instance.

InstName
Specifies the name of the instance which is also the name of an existing user in the operating system.

For Windows operating systems

InstName
Specifies the name of the instance.

-s *InstType*
Specifies the type of instance to create. Valid values are:

client Used to create an instance for a client. Use this value if you are using DB2 Connect Personal Edition.

standalone
Used to create an instance for a database server with local clients.

ese Used to create an instance for a database server with local and remote clients.

wse Used to create an instance for DB2 Workgroup Server Edition, DB2 Express Edition and DB2 Connect Enterprise Edition.

-u *Username, Password*
Specifies the account name and password for the DB2 service. This option is required when creating a partitioned database instance.

-p *InstProfPath*
Specifies the instance profile path.

-h *HostName*
Overrides the default TCP/IP host name if there is more than one for the current machine. The TCP/IP host name is used when creating the default database partition (database partition 0). This option is only valid for partitioned database instances.

-r *PortRange*
Specifies a range of TCP/IP ports to be used by the partitioned database instance when running in MPP mode. For example, `-r 50000,50007`. The services file of the local machine will be updated with the following entries if this option is specified:

```
DB2_InstName         baseport/tcp
DB2_InstName_END     endport/tcp
```

-? Displays usage information.

Examples:

- On an AIX machine, to create an instance for the user ID `db2inst1`, issue the following command:

  On a client machine:

  ```
  DB2DIR/instance/db2icrt db2inst1
  ```

  On a server machine:

  ```
  DB2DIR/instance/db2icrt -u db2fenc1 db2inst1
  ```

  where `db2fenc1` is the user ID under which fenced user-defined functions and fenced stored procedures will run.

Usage notes:

- The -s option is intended for situations in which you want to create an instance that does not use the full functionality of the system. For example, if you are using Enterprise Server Edition (ESE), but do not want partition capabilities, you could create a Workgroup Server Edition (WSE) instance, using the option `-s WSE`.

- To create a DB2 instance that supports Microsoft Cluster Server, first create an instance, then use the `db2iclus` command to migrate it to run in a MSCS instance.
- Only once instance can be created under a user name. If you want to create an instance under a user name that already has a related instance, you must drop instance before creating the new one.

Related concepts:

- "User, user ID and group naming rules" on page 26

Related reference:

- "db2iclus - Microsoft cluster server command" in *Command Reference*

db2iupdt - Update instances

On Linux and UNIX-bsed systems, this command updates a specified DB2 instance. The `db2iupdt` command can be issued against instances of the same version that are assoicated with the same, or a different DB2 database installation directory. In all cases, it will update the instance so that it runs against the code located in the same directory as where you issued the `db2iupdt` command. You should issue this command:

- whenever you install a new DB2 database product or Fix Pack to the installation directory related to the DB2 instance.
- if you want to bring a DB2 instance from one installation path to the current one for the same version of DB2 database system.

On Linux and UNIX-based systems, it is located in the `DB2DIR/instance` directory, where `DB2DIR` is the location where the current version of the DB2 database product is installed.

On Windows operating systems, this command updates the instance release level. It can also be used to move an instance from one DB2 copy to another. The instance is moved to the DB2 copy you execute `db2iupdt` from. To move your instance profile from its current location to another location, use the `/p` option and specify the instance profile path. Otherwise, the instance profile will stay in its original location after update. Use the `db2imigr` command instead to change from a major release to another. This utility is located in the `DB2PATH\sqllib\bin` directory, where `DB2PATH` is the location where the current version of the DB2 database product is installed.

To update an instance with `db2iupdt`, you must first stop all processes that are running for the instance.

Authorization:

Root access on UNIX operating systems or Local Administrator on Windows operating systems.

Command syntax:

For UNIX operating systems

```
>>─db2iupdt──┬──────┬──┬────┬──┬────┬──┬────┬──┬────┬──┬──────────────┬──────────►
             ├─-h───┤  └─-d─┘  └─-k─┘  └─-D─┘  └─-s─┘  └─-a──AuthType─┘
             └─-?───┘
```

```
►─┬──────────────┬─┬─InstName─┬──────────────────────────────────────────────►◄
  └─-u──FencedID─┘ └─-e───────┘
```

For Windows operating systems

```
►►──db2iupdt──InstName──/u:──username,password──┬────────────────────────────┬──►
                                                └─/p:──instance-profile-path─┘

►──┬──────────────────────────┬──┬───────────────┬──┬────┬──┬────┬──────────────►
   └─/r:──baseport,endport────┘  └─/h:──hostname─┘  └─/s─┘  └─/q─┘

►──┬─────────────────┬──┬────┬──────────────────────────────────────────────────►◄
   └─/a:──authType───┘  └─/?─┘
```

Command parameters:
For UNIX operating systems

-h or -?
Displays the usage information.

-d Turns debug mode on.

-k Keeps the current instance type during the update.

-D Moves an instance from a higher code level on one path to a lower code level installed on another path.

-s Ignores the existing SPM log directory.

-a *AuthType*
Specifies the authentication type (SERVER, SERVER_ENCRYPT or CLIENT) for the instance. The default is SERVER.

-u *Fenced ID*
Specifies the name of the user ID under which fenced user defined functions and fenced stored procedures will run. This option is only needed when converting an instance from a client instance to a server instance type. If an instance is already a server instance, or if an instance is a client instance and is staying as a client instance (by using the `-k` option), the `-u` option is not needed. The `-u` option cannot change the fenced user for an existing instance.

InstName
Specifies the name of the instance.

-e Updates every instance.

For Windows operating systems

InstName
Specifies the name of the instance.

/u:*username,password*
Specifies the account name and password for the DB2 service.

/p:*instance-profile-path*
Specifies the new instance profile path for the updated instance.

/r:*baseport,endport*
Specifies the range of TCP/IP ports to be used by the partitioned database

instance when running in MPP mode. When this option is specified, the services file on the local machine will be updated with the following entries:

```
DB2_InstName       baseport/tcp
DB2_InstName_END   endport/tcp
```

/h:*hostname*
Overrides the default TCP/IP host name if there are more than one TCP/IP host names for the current machine.

/s Updates the instance to a partitioned instance.

/q Issues the db2iupdt command in quiet mode.

/a:*authType*
Specifies, authType, the authentication type (SERVER, CLIENT, or SERVER_ENCRYPT) for the instance .

/? Displays usage information for the db2iupdt command.

Examples (UNIX):

1. If you have an instance db2inst1 related to the installation path DB2DIR and you applied a Fix Pack on top of the installation path, you may need to update the instance by running the following command:
   ```
   <DB2DIR>/instance/db2iupdt db2inst1
   ```
   This will bring up the instance to the highest type of instance. To keep the original instance type, you might need to use the -k option.
2. An instance, db2inst2, is related to the installation path DB2DIR1. You have another installation of the DB2 database product on the same system at DB2DIR2 for the same version of the DB2 database product as that installed on DB2DIR1. To update the instance to use the installed DB2 database product from DB2DIR1 to DB2DIR2, issue the following command:
   ```
   <DB2DIR2>/instance/db2iupdt db2inst2
   ```
   If the DB2 database product installed at DB2DIR2 is at level lower than that at DB2DIR1, issue:
   ```
   <DB2DIR2>/instance/db2iupdt -D db2inst2
   ```

Usage notes:

For UNIX operating systems

- If you use the db2iupdt command to update a DB2 instance from another installation location to the current installation location, the DB2 Global Profile Variables defined in an old DB2 database installation path will not be updated over to the new installation location. The DB2 Instance Profile Varibles specific to the instance to be updated will be carried over after the instance is updated.

Related tasks:

- "Updating instance configuration on Windows" in *Administration Guide: Implementation*
- "Updating instance configuration on UNIX" in *Administration Guide: Implementation*

Related reference:

- "db2ilist - List instances command" in *Command Reference*

db2nchg - Change database partition server configuration

Modifies database partition server configuration. This includes moving the database partition server (node) from one machine to another; changing the TCP/IP host name of the machine; and selecting a different logical port number or a different network name for the database partition server (node). This command can only be used if the database partition server is stopped.

This command is available on Windows-based operating systems only.

Authorization:

Local Administrator

Command syntax:

```
►►──db2nchg──/n:──dbpartitionnum──┬──────────────────────┬──────────────────────►
                                  └─/i:──instance_name───┘

►─┬────────────────────────┬──┬─────────────────────┬──┬───────────────────┬────►
  └─/u:──user,password─────┘  └─/p:──logical_port───┘  └─/h:──hostname─────┘

►─┬──────────────────────┬──┬──────────────────────┬─────────────────────────►◄
  └─/m:──machine_name────┘  └─/g:──network_name────┘
```

Command parameters:

/n:dbpartitionnum
: Specifies the database partition number of the database partition server's configuration that is to be changed.

/i:instance_name
: Specifies the instance in which this database partition server participates. If a parameter is not specified, the default is the current instance.

/u:username,password
: Specifies the user name and password. If a parameter is not specified, the existing user name and password will apply.

/p:logical_port
: Specifies the logical port for the database partition server. This parameter must be specified to move the database partition server to a different machine. If a parameter is not specified, the logical port number will remain unchanged.

/h:host_name
: Specifies TCP/IP host name used by FCM for internal communications. If this parameter is not specified, the host name will remain the same.

/m:machine_name
: Specifies the machine where the database partition server will reside. The database partition server can only be moved if there are no existing databases in the instance.

/g:network_name
: Changes the network name for the database partition server. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a machine. The network name or the IP address can be entered.

Examples:

To change the logical port assigned to database partition 2, which participates in the instance TESTMPP, to logical port 3, enter the following command:

```
db2nchg /n:2 /i:TESTMPP /p:3
```

Related reference:

- "db2ncrt - Add database partition server to an instance " on page 486
- "db2ndrop - Drop database partition server from an instance " on page 487

db2ncrt - Add database partition server to an instance

Adds a database partition server (node) to an instance.

This command is available on Windows operating systems only.

Scope:

If a database partition server is added to a computer where an instance already exists, a database partition server is added as a logical database partition server to the computer. If a database partition server is added to a computer where an instance does not exist, the instance is added and the computer becomes a new physical database partition server. This command should not be used if there are databases in an instance. Instead, the START DATABASE MANAGER command should be issued with the ADD DBPARTITIONNUM option. This ensures that the database is correctly added to the new database partition server. It is also possible to add a database partition server to an instance in which a database has been created. The db2nodes.cfg file should not be edited since changing the file might cause inconsistencies in the partitioned database system.

Authorization:

Local Administrator authority on the computer where the new database partition server is added.

Command syntax:

```
►►──db2ncrt──/n:──dbpartitionnum──/u:──username,password─────────────────────►

►─┬──────────────────┬──┬─────────────────┬──┬─────────────────┬────────────►
  └─/i:──instance_name─┘  └─/m:──machine_name─┘  └─/p:──logical_port─┘

►─┬──────────────┬──┬─────────────────┬──┬──────────────────────────────┬───►◄
  └─/h:──host_name─┘  └─/g:──network_name─┘  └─/o:──instance_owning_machine─┘
```

Command parameters:

/n:dbpartitionnum
: A unique database partition number which identifies the database partition server. The number entered can range from 1 to 999.

/u:domain_name\username,password
: Specifies the domain, logon account name and password for DB2.

/i:instance_name
: Specifies the instance name. If a parameter is not specified, the default is the current instance.

/m:machine_name
Specifies the computer name of the Windows workstation on which the database partition server resides. This parameter is required if a database partition server is added on a remote computer.

/p:logical_port
Specifies the logical port number used for the database partition server. If this parameter is not specified, the logical port number assigned will be 0. When creating a logical database partition server, this parameter must be specified and a logical port number that is not in use must be selected. Note the following restrictions:

- Every computer must have a database partition server that has a logical port 0.
- The port number cannot exceed the port range reserved for FCM communications in the x:\winnt\system32\drivers\etc\ directory. For example, if a range of 4 ports is reserved for the current instance, then the maximum port number is 3. Port 0 is used for the default logical database partition server.

/h:host_name
Specifies the TCP/IP host name that is used by FCM for internal communications. This parameter is required when the database partition server is being added on a remote computer.

/g:network_name
Specifies the network name for the database partition server. If a parameter is not specified, the first IP address detected on the system will be used. This parameter can be used to apply a specific IP address to the database partition server when there are multiple IP addresses on a computer. The network name or the IP address can be entered.

/o:instance_owning_machine
Specifies the computer name of the instance-owning computer. The default is the local computer. This parameter is required when the **db2ncrt** command is invoked on any computer that is not the instance-owning computer.

Examples:

To add a new database partition server to the instance TESTMPP on the instance-owning computer SHAYER, where the new database partition server is known as database partition 2 and uses logical port 1, enter the following command:

```
db2ncrt /n:2 /u:QBPAULZ\paulz,g1reeky /i:TESTMPP /m:TEST /p:1 /o:SHAYER
```

Related reference:

- "db2nchg - Change database partition server configuration " on page 485
- "db2ndrop - Drop database partition server from an instance " on page 487

db2ndrop - Drop database partition server from an instance

Drops a database partition server (node) from an instance that has no databases. If a database partition server is dropped, its database partition number can be reused for a new database partition server. This command can only be used if the database partition server is stopped.

db2ndrop - Drop Database Partition Server from an Instance

This command is available on Windows-based operating systems only.

Authorization:

Local Administrator authority on the machine where the database partition server is being dropped.

Command syntax:

```
►►──db2ndrop──/n:──dbpartitionnum──┬─────────────────────┬────────────►◄
                                   └─/i:──instance_name─┘
```

Command parameters:

/n:dbpartitionnum
A unique database partition number which identifies the database partition server.

/i:instance_name
Specifies the instance name. If a parameter is not specified, the default is the current instance.

Examples:

```
db2ndrop /n:2 /i=KMASCI
```

Usage notes:

If the instance-owning database partition server (dbpartitionnum 0) is dropped from the instance, the instance becomes unusable. To drop the instance, use the **db2idrop** command.

This command should not be used if there are databases in this instance. Instead, the `db2stop drop nodenum` command should be used. This ensures that the database partition server is correctly removed from the partition database system. It is also possible to drop a database partition server in an instance where a database exists. The db2nodes.cfg file should not be edited since changing the file might cause inconsistencies in the partitioned database system.

To drop a database partition server that is assigned to the logical port 0 from a machine that is running multiple logical database partition servers, all other database partition servers assigned to the other logical ports must be dropped first. Each database partition server must have a database partition server assigned to logical port 0.

Related reference:

- “db2nchg - Change database partition server configuration ” on page 485
- “db2ncrt - Add database partition server to an instance ” on page 486

db2rbind - Rebind all packages

Rebinds packages in a database.

Authorization:

One of the following:
- *sysadm*

Required connection:

None

Command syntax:

```
►►──db2rbind──database──-l──logfile──┬─────┬──┬──────────────────────────┬──────────►
                                     └─all─┘  └─-u──userid──-p──password─┘

►──┬──────────────────────────┬─────────────────────────────────────────────────────►◄
   │       ┌─conservative─┐   │
   └─-r────┴─any──────────┴───┘
```

Command parameters:

database
Specifies an alias name for the database whose packages are to be revalidated.

-l
Specifies the (optional) path and the (mandatory) file name to be used for recording errors that result from the package revalidation procedure.

all
Specifies that rebinding of all valid and invalid packages is to be done. If this option is not specified, all packages in the database are examined, but only those packages that are marked as invalid are rebound, so that they are not rebound implicitly during application execution.

-u
User ID. This parameter must be specified if a password is specified.

-p
Password. This parameter must be specified if a user ID is specified.

-r
Resolve. Specifies whether rebinding of the package is to be performed with or without conservative binding semantics. This affects whether new functions and data types are considered during function resolution and type resolution on static DML statements in the package. This option is not supported by DRDA. Valid values are:

conservative
Only functions and types in the SQL path that were defined before the last explicit bind time stamp are considered for function and type resolution. Conservative binding semantics are used. This is the default. This option is not supported for an inoperative package.

any
Any of the functions and types in the SQL path are considered for function and type resolution. Conservative binding semantics are not used.

Usage notes:

- This command uses the rebind API (`sqlarbnd`) to attempt the revalidation of all packages in a database.
- Use of **db2rbind** is not mandatory.
- For packages that are invalid, you can choose to allow package revalidation to occur implicitly when the package is first used. You can choose to selectively revalidate packages with either the `REBIND` or the `BIND` command.
- If the rebind of any of the packages encounters a deadlock or a lock timeout the rebind of all the packages will be rolled back.

Related reference:
- "BIND" on page 441
- "PRECOMPILE " on page 635
- "REBIND " on page 659

db2extsec - Set permissions for DB2 objects

Sets the permissions for DB2 objects (for example, files, directories, network shares, registry keys and services) on updated DB2 database system installations. In previous releases, this command was named `db2secv82`. The command name `db2secv82` is deprecated but can be used as an alternative name for `db2extsec`.

Authorization:
- *sysadm*

Required connection:

none

Command syntax:

```
>>—db2extsec——————————————————————————————————————————————————————————><
              └─/u—usergroup─┘  └─/a—admingroup─┘  └─/r─┘
```

Command parameters:

/u *usergroup*
: Specifies the name of the user group to be added. If this option is not specified, the default DB2 user group (DB2USERS) is used.

/a *admingroup*
: Specifies the name of the administration group to be added. If this option is not specified, the default DB2 administration group (DB2ADMNS) is used.

/r
: Specifies that the changes made by previously running `db2extsec` should be reversed. If you specify this option, all other options are ignored. This option will only work if no other DB2 commands have been issued since the `db2extsec` command was issued.

Related concepts:
- Chapter 18, "Extended Windows security using DB2ADMNS and DB2USERS groups," on page 153

Related tasks:

- “Adding your user ID to the DB2ADMNS and DB2USERS user groups (Windows)” in *Quick Beginnings for DB2 Servers*

db2set - DB2 profile registry

Displays, sets, or removes DB2 profile variables. An external environment registry command that supports local and remote administration, via the DB2 Administration Server, of DB2’s environment variables stored in the DB2 profile registry.

Authorization:

sysadm

Required connection:

None

Command syntax:

```
►►──db2set──┬───────────────────┬──┬────────────────────────────────────┬──────────►
            └─variable=──value──┘  ├─-g─────────────────────────────────┤
                                   ├─-i──instance──┬──────────────────────┬┤
                                   │               └─db-partition-number─┘│
                                   └─-gl─────────────────────────────────┘

►──┬──────┬──┬───────┬──┬─-r──┬────────────┬──┬─────────────────────┬─┬──────────►
   └─-all─┘  └─-null─┘  │     └─instance──┘  └─db-partition-number─┘ │
                        └─────────────────────────────────────────────┘

►──┬─────────────────────────────────────────────┬──┬─────┬──┬────┬──┬─────┬──────►
   └─-n──DAS node──┬──────────────────────────┬──┘  ├─-l──┤  └─-v─┘  ├─-ul─┤
                   └─-u──user──┬─────────────┬┘     └─-lr─┘          └─-ur─┘
                               └─-p──password─┘

►──┬────┬─────────────────────────────────────────────────────────────────────►◄
   ├─-h─┤
   └─-?─┘
```

Command parameters:

variable= value
: Sets a specified variable to a specified value. To delete a variable, do not specify a value for the specified variable. Changes to settings take effect after the instance has been restarted.

-g
: Accesses the global profile registry variables for all instances pertaining to a particular DB2 copy.

-i
: Specifies the instance profile to use instead of the current, or default.

db-partition-number
: Specifies a number listed in the `db2nodes.cfg` file.

-gl
: Accesses the global profile variables stored in LDAP. This option is only effective if the registry variable DB2_ENABLE_LDAP has been set to YES.

-all
: Displays all occurrences of the local environment variables as defined in:
- The environment, denoted by [e]

- The node level registry, denoted by [n]
- The instance level registry, denoted by [i]
- The global level registry, denoted by [g].

-null Sets the value of the variable at the specified registry level to NULL. This avoids having to look up the value in the next registry level, as defined by the search order.

-r instance
Resets the profile registry for the given instance. If no instance is specified, and an instance attachment exists, resets the profile for the current instance. If no instance is specified, and no attachment exists, resets the profile for the instance specified by the DB2INSTANCE environment variable.

-n DAS node
Specifies the remote DB2 administration server node name.

-u user
Specifies the user ID to use for the administration server attachment.

-p password
Specifies the password to use for the administration server attachment.

-l Lists all instance profiles for the current DB2 product installation.

-lr Lists all supported registry variables.

-v Specifies verbose mode.

-ul Accesses the user profile variables. This parameter is supported on Windows operating systems only.

-ur Refreshes the user profile variables. This parameter is supported on Windows operating systems only.

-h/-? Displays help information. When this option is specified, all other options are ignored, and only the help information is displayed.

Examples:

- Display all defined profiles (DB2 instances) pertaining to a particular installation :
  ```
  db2set -l
  ```
- Display all supported registry variables:
  ```
  db2set -lr
  ```
- Display all defined global variables which are visible by all instances pertaining to a particular installation:
  ```
  db2set -g
  ```
- Display all defined variables for the current instance:
  ```
  db2set
  ```
- Display all defined values for the current instance:
  ```
  db2set -all
  ```
- Display all defined values for DB2COMM for the current instance:
  ```
  db2set -all DB2COMM
  ```
- Reset all defined variables for the instance INST on node 3:
  ```
  db2set -r -i INST 3
  ```
- Unset the variable DB2CHKPTR on the remote instance RMTINST through the DAS node RMTDAS using user ID MYID and password MYPASSWD:

```
db2set -i RMTINST -n RMTDAS -u MYID -p MYPASSWD DB2CHKPTR=
```

- Set the variable DB2COMM to be TCPIP globally for all instances pertaining to a particular installation:
  ```
  db2set -g DB2COMM=TCPIP
  ```
- Set the variable DB2COMM to be only TCPIP for instance MYINST:
  ```
  db2set -i MYINST DB2COMM=TCPIP
  ```
- Set the variable DB2COMM to null at the given instance level:
  ```
  db2set -null DB2COMM
  ```

Usage notes:

If no variable name is specified, the values of all defined variables are displayed. If a variable name *is* specified, only the value of that variable is displayed. To display all the defined values of a variable, specify *variable* `-all`. To display all the defined variables in all registries, specify `-all`.

To modify the value of a variable, specify *variable*`=`, followed by its new value. To set the value of a variable to NULL, specify *variable* `-null`. Changes to settings take effect after the instance has been restarted.

To delete a variable, specify *variable*`=`, followed by no value.

Related reference:

- "REG_VARIABLES administrative view - Retrieve DB2 registry settings in use" in *Administrative SQL Routines and Views*

db2undgp - Revoke execute privilege

Revoke the execute privilege on external stored procedures. This command can be used against external stored procedures.

During the database migration, EXECUTE for all existing functions, methods, and External stored procedure is granted to PUBLIC. This will cause a security exposure for External Stored procedures that contain SQL data access. To prevent users from accessing SQL objects which the user might not have privilege for, use the db2undgp command.

Command syntax:

```
►►—db2undgp—┬──────────────┬─┬────┬─┬───────────────┬─┬────┬──────────────►◄
            └─-d—dbname────┘ └─-h─┘ └─-o—outfile────┘ └─-r─┘
```

Command parameters:

-d *dbname*
: Specifies a database name whose maximum length is 8 characters.

-h
: Displays help for the command.

-o *outfile*
: Output the revoke statements in the specified file. Length of the File name should be <= 80.

-r
: Perform the revoke

Usage notes:

At least one of the -r or -o options must be specified.

Related reference:

- "Changes to the EXECUTE privilege on PUBLIC for migrated routines" in *Migration Guide*

DROP DATABASE

Deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.

Scope:

By default, this command affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*

Required connection:

Instance. An explicit attachment is not required. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the command.

Command syntax:

```
►►──DROP──┬─DATABASE─┬──database-alias──┬──────────────────┬──────────────►◄
          └─DB───────┘                  └─AT DBPARTITIONNUM─┘
```

Command parameters:

DATABASE database-alias
: Specifies the alias of the database to be dropped. The database must be cataloged in the system database directory.

AT DBPARTITIONNUM
: Specifies that the database is to be deleted only on the database partition that issued the DROP DATABASE command. This parameter is used by utilities supplied with DB2 ESE, and is not intended for general use. Improper use of this parameter can cause inconsistencies in the system, so it should only be used with caution.

Examples:

The following example deletes the database referenced by the database alias SAMPLE:

```
db2 drop database sample
```

Usage notes:

DROP DATABASE deletes all user data and log files, as well as any back/restore history for the database. If the log files are needed for a roll-forward recovery after a restore operation, or the backup history required to restore the database, these files should be saved prior to issuing this command.

The database must not be in use; all users must be disconnected from the database before the database can be dropped.

To be dropped, a database must be cataloged in the system database directory. Only the specified database alias is removed from the system database directory. If other aliases with the same database name exist, their entries remain. If the database being dropped is the last entry in the local database directory, the local database directory is deleted automatically.

If DROP DATABASE is issued from a remote client (or from a different instance on the same machine), the specified alias is removed from the client's system database directory. The corresponding database name is removed from the server's system database directory.

This command unlinks all files that are linked through any DATALINK columns. Since the unlink operation is performed asynchronously on the DB2 Data Links Manager, its effects might not be seen immediately on the DB2 Data Links Manager, and the unlinked files might not be immediately available for other operations. When the command is issued, all the DB2 Data Links Managers configured to that database must be available; otherwise, the drop database operation will fail.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODE can be substituted for DBPARTITIONNUM.

Related tasks:
- "Dropping a database" in *Administration Guide: Implementation*

Related reference:
- "CREATE DATABASE " on page 461
- "UNCATALOG DATABASE command" in *Command Reference*
- "sqledpan - Drop a database on a database partition server" on page 807
- "sqledrpd - Drop database" on page 808
- "CATALOG DATABASE " on page 458

DROP DBPARTITIONNUM VERIFY

Verifies if a database partition exists in the database partition groups of any databases, and if an event monitor is defined on the database partition. This command should be used prior to dropping a database partition from a partitioned database system.

Scope:

This command only affects the database partition on which it is issued.

Authorization:

sysadm

Command syntax:

```
►►──DROP DBPARTITIONNUM VERIFY──────────────────────────────────────►◄
```

Command parameters:
None
Usage notes:
If a message is returned, indicating that the database partition is not in use, use the STOP DATABASE MANAGER command with DROP DBPARTITIONNUM to remove the entry for the database partition from the `db2nodes.cfg` file, which removes the database partition from the database system.

If a message is returned, indicating that the database partition is in use, the following actions should be taken:

1. If the database partition contains data, redistribute the data to remove it from the database partition using REDISTRIBUTE DATABASE PARTITION GROUP. Use either the DROP DBPARTITIONNUM option on the REDISTRIBUTE DATABASE PARTITION GROUP command or on the ALTER DATABASE PARTITION GROUP statement to remove the database partition from any database partition groups for the database. This must be done for each database that contains the database partition in a database partition group.
2. Drop any event monitors that are defined on the database partition.
3. Rerun DROP DBPARTITIONNUM VERIFY to ensure that the database is no longer in use.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODE can be substituted for DBPARTITIONNUM.

Related reference:

- "STOP DATABASE MANAGER " on page 624
- "REDISTRIBUTE DATABASE PARTITION GROUP " on page 577
- "sqledrpn - Check whether a database partition server can be dropped" on page 810

EXPORT

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Authorization:

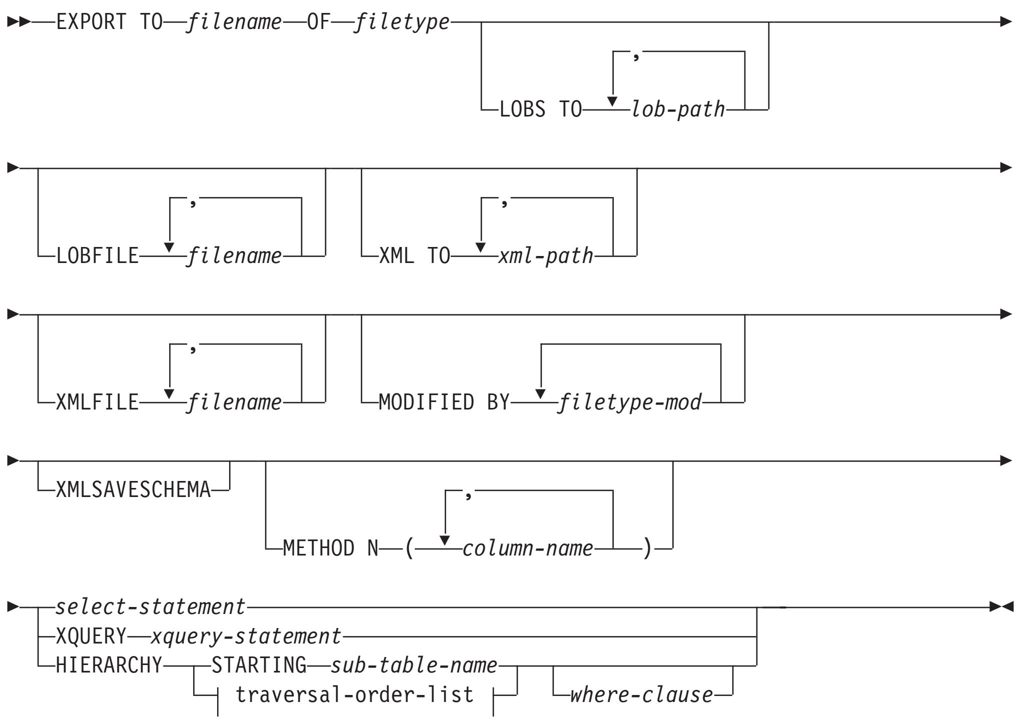
One of the following:

- *sysadm*
- *dbadm*

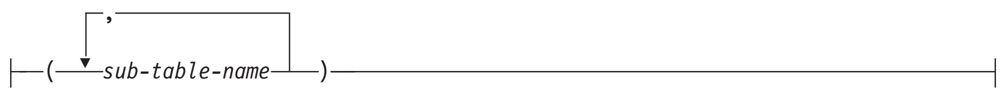
or CONTROL or SELECT privilege on each participating table or view.

Required connection:

Command syntax:



traversal-order-list:



Command parameters:

HIERARCHY traversal-order-list
: Export a sub-hierarchy using the specified traverse order. All sub-tables must be listed in PRE-ORDER fashion. The first sub-table name is used as the target table name for the SELECT statement.

HIERARCHY STARTING sub-table-name
: Using the default traverse order (OUTER order for ASC, DEL, or WSF files, or the order stored in PC/IXF data files), export a sub-hierarchy starting from *sub-table-name*.

LOBFILE filename
: Specifies one or more base file names for the LOB files. When name space is exhausted for the first name, the second name is used, and so on. The maximum number of file names that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

  When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from *lob-path*), and then appending a 3-digit sequence number and the three character identifier `lob`. For example, if the current LOB path is the directory `/u/foo/lob/path/`, and the current LOB file name is `bar`, the LOB files created will be `/u/foo/lob/path/bar.001.lob`, `/u/foo/lob/path/bar.002.lob`, and so on.

LOBS TO lob-path
Specifies one or more paths to directories in which the LOB files are to be stored. There will be at least one file per LOB path, and each file will contain at least one LOB. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behavior.

METHOD N column-name
Specifies one or more column names to be used in the output file. If this parameter is not specified, the column names in the table are used. This parameter is valid only for WSF and IXF files, but is not valid when exporting hierarchical data.

MODIFIED BY filetype-mod
Specifies file type modifier options. See File type modifiers for the export utility.

OF filetype
Specifies the format of the data in the output file:

- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs.
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony

  When exporting BIGINT or DECIMAL data, only values that fall within the range of type DOUBLE can be exported accurately. Although values that do not fall within this range are also exported, importing or loading these values back might result in incorrect data, depending on the operating system.
- IXF (integrated exchange format, PC version), in which most of the table attributes, as well as any existing indexes, are saved in the IXF file, except when columns are specified in the SELECT statement. With this format, the table can be recreated, while with the other file formats, the table must already exist before data can be imported into it.

select-statement
Specifies the SELECT or XQUERY statement that will return the data to be exported. If the statement causes an error, a message is written to the message file (or to standard output). If the error code is one of SQL0012W, SQL0347W, SQL0360W, SQL0437W, or SQL1824W, the export operation continues; otherwise, it stops.

TO filename

If the name of a file that already exists is specified, the export utility overwrites the contents of the file; it does not append the information.

XMLFILE filename
Specifies one or more base file names for the XML files. When name space is exhausted for the first name, the second name is used, and so on.

When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from xml-path), appending a 3-digit sequence number, and appending the three character identifier `xml`. For example, if the current XML path is the directory `/u/foo/xml/path/`, and the current XML file name is `bar`, the XML files created will be `/u/foo/xml/path/bar.001.xml`, `/u/foo/xml/path/bar.002.xml`, and so on.

XML TO xml-path
: Specifies one or more paths to directories in which the XML files are to be stored. There will be at least one file per XML path, and each file will contain at least one XQuery Data Model (QDM) instance. If more than one path is specified, then QDM instances are distributed evenly among the paths.

XMLSAVESCHEMA
: Specifies that XML schema information should be saved for all XML columns. For each exported XML document that was validated against an XML schema when it was inserted, the fully qualified SQL identifier of that schema will be stored as an (SCH) attribute inside the corresponding XML Data Specifier (XDS). If the exported document was not validated against an XML schema or the schema object no longer exists in the database, an SCH attribute will not be included in the corresponding XDS.

  The schema and name portions of the SQL identifier are stored as the ″OBJECTSCHEMA″ and ″OBJECTNAME″ values in the row of the SYSCAT.XSROBJECTS catalog table corresponding to the XML schema.

  The `XMLSAVESCHEMA` option is not compatible with XQuery sequences that do not produce well-formed XML documents.

Usage notes:

- Be sure to complete all table operations and release all locks before starting an export operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.
- Table aliases can be used in the SELECT statement.
- The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.
- The export utility produces a warning message whenever a character column with a length greater than 254 is selected for export to DEL format files.
- PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.
- The file copying step is not necessary if the source and the target databases are both accessible from the same client.
- DB2 Connect can be used to export tables from DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF export is supported.
- The export utility will not create multiple-part PC/IXF files when invoked from an AIX system.
- The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form `SELECT * FROM tablename`.
- When exporting typed tables, subselect statements can only be expressed by specifying the target table name and the WHERE clause. Fullselect and *select-statement* cannot be specified when exporting a hierarchy.
- For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.

- Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.
- When exporting data from a table that has protected rows, the LBAC credentials held by the session authorization id might limit the rows that are exported. Rows that the session authorization ID does not have read access to will not be exported. No error or warning is given.
- If the LBAC credentials held by the session authorization id do not allow reading from one or more protected columns included in the export then the export fails and an error (SQLSTATE 42512) is returned.
- Export packages are bound using `DATETIME ISO` format, thus, all date/time/timestamp values are converted into ISO format when cast to a string representation. Since the CLP packages are bound using `DATETIME LOC` format (locale specific format), you may see inconsistant behaviour between CLP and export if the CLP DATETIME format is different from ISO. For instance, the following SELECT statement may return expected results:

  ```
  db2 select col2 from tab1 where char(col2)='05/10/2005';
     COL2
     ----------
     05/10/2005
     05/10/2005
     05/10/2005
     3 record(s) selected.
  ```

  But an export command using the same select clause will not:

  ```
  db2 export to test.del of del select col2 from test
  where char(col2)='05/10/2005';
     Number of rows exported: 0
  ```

  Now, replacing the LOCALE date format with ISO format gives the expected results:

  ```
  db2 export to test.del of del select col2 from test
  where char(col2)='2005-05-10';
     Number of rows exported: 3
  ```

Related concepts:
- "Export Overview" in *Data Movement Utilities Guide and Reference*
- "Privileges, authorities and authorization required to use export" on page 413

Related tasks:
- "Exporting data" in *Data Movement Utilities Guide and Reference*

Related reference:
- "ADMIN_CMD procedure ÔÇô Run administrative commands" in *Administrative SQL Routines and Views*
- "EXPORT command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "Export Sessions - CLP Examples" in *Data Movement Utilities Guide and Reference*
- "LOB and XML file behavior with regard to import and export" in *Data Movement Utilities Guide and Reference*

GET AUTHORIZATIONS

Reports the authorities of the current user from values found in the database configuration file and the authorization system catalog view (SYSCAT.DBAUTH).

Authorization:

None

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax:

```
►►──GET AUTHORIZATIONS──────────────────────────────────────────────────►◄
```

Command parameters:
None
Examples:
The following is sample output from GET AUTHORIZATIONS:

```
Administrative Authorizations for Current User

Direct SYSADM authority                    = NO
Direct SYSCTRL authority                   = NO
Direct SYSMAINT authority                  = NO
Direct DBADM authority                     = YES
Direct CREATETAB authority                 = YES
Direct BINDADD authority                   = YES
Direct CONNECT authority                   = YES
Direct CREATE_NOT_FENC authority           = YES
Direct IMPLICIT_SCHEMA authority           = YES
Direct LOAD authority                      = YES
Direct QUIESCE_CONNECT authority           = YES
Direct CREATE_EXTERNAL_ROUTINE authority   = YES

Indirect SYSADM authority                  = YES
Indirect SYSCTRL authority                 = NO
Indirect SYSMAINT authority                = NO
Indirect DBADM authority                   = NO
Indirect CREATETAB authority               = YES
Indirect BINDADD authority                 = YES
Indirect CONNECT authority                 = YES
Indirect CREATE_NOT_FENC authority         = NO
Indirect IMPLICIT_SCHEMA authority         = YES
Indirect LOAD authority                    = NO
Indirect QUIESCE_CONNECT authority         = NO
Indirect CREATE_EXTERNAL_ROUTINE authority = NO
```

Usage notes:

- Direct authorities are acquired by explicit commands that grant the authorities to a user ID. Indirect authorities are based on authorities acquired by the groups to which a user belongs. (All users belong to a special group called PUBLIC)
- The GET AUTHORIZATIONS command does not display whether or not the current user holds SECADM authority. To find out who holds SECADM authority, use the following query:

```
SELECT GRANTEE FROM SYSCAT.DBAUTH
WHERE SECURITYADMAUTH = 'Y'
```

Related concepts:

- "Authorization" on page 62

Related reference:

- "SYSCAT.DBAUTH " on page 173

GET DATABASE CONFIGURATION

Returns the values of individual entries in a specific database configuration file.

Scope:

This command returns information only for the database partition on which it is executed.

Authorization:

None

Required connection:

Instance. An explicit attachment is not required, but a connection to the database is required when using the SHOW DETAIL clause. If the database is listed as remote, an instance attachment to the remote node is established for the duration of the command.

Command syntax:

```
►►──GET──┬─DATABASE─┬──┬─CONFIGURATION─┬──┬──────────────────────────┬──────────►
         └─DB───────┘  ├─CONFIG────────┤  └─FOR──database-alias──────┘
                       └─CFG───────────┘

►──┬─────────────┬──────────────────────────────────────────────────────────►◄
   └─SHOW DETAIL─┘
```

Command parameters:

FOR database-alias
: Specifies the alias of the database whose configuration is to be displayed. You do not need to specify the alias if a connection to the database already exists.

SHOW DETAIL
: Displays detailed information showing the current value of database configuration parameters as well as the value of the parameters the next time you activate the database. This option lets you see the result of dynamic changes to configuration parameters.

Examples:

Notes:

1. Output on different platforms might show small variations reflecting platform-specific parameters.

2. Parameters with keywords enclosed by parentheses can be changed by the UPDATE DATABASE CONFIGURATION command.
3. Fields that do not contain keywords are maintained by the database manager and cannot be updated.

The following is sample output from GET DATABASE CONFIGURATION (issued on AIX):

```
                  Database Configuration for Database mick

 Database configuration release level                    = 0x0a00
 Database release level                                  = 0x0a00

 Database territory                                      = en_US
 Database code page                                      = 819
 Database code set                                       = ISO8859-1
 Database country/region code                            = 1
 Database collating sequence                             = UNIQUE
 Alternate collating sequence              (ALT_COLLATE) =
 Database page size                                      = 4096
 Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE

 Discovery support for this database       (DISCOVER_DB) = ENABLE

 Default query optimization class         (DFT_QUERYOPT) = 5
 Degree of parallelism                      (DFT_DEGREE) = 1
 Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO
 Default refresh age                   (DFT_REFRESH_AGE) = 0
 Default maintained table types for opt (DFT_MTTB_TYPES) = SYSTEM
 Number of frequent values retained     (NUM_FREQVALUES) = 10
 Number of quantiles retained            (NUM_QUANTILES) = 20

 Backup pending                                          = NO

 Database is consistent                                  = YES
 Rollforward pending                                     = NO
 Restore pending                                         = NO

 Multi-page file allocation enabled                      = YES

 Log retain for recovery status                          = NO
 User exit for logging status                            = NO

 Data Links Token Expiry Interval (sec)      (DL_EXPINT) = 60
 Data Links Write Token Init Expiry Intvl(DL_WT_IEXPINT) = 60
 Data Links Number of Copies             (DL_NUM_COPIES) = 1
 Data Links Time after Drop (days)        (DL_TIME_DROP) = 1
 Data Links Token in Uppercase                (DL_UPPER) = NO
 Data Links Token Algorithm                   (DL_TOKEN) = MAC0

 Database heap (4KB)                            (DBHEAP) = 1200
 Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC
 Catalog cache size (4KB)              (CATALOGCACHE_SZ) = 64
 Log buffer size (4KB)                        (LOGBUFSZ) = 8
 Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000
 Buffer pool size (pages)                     (BUFFPAGE) = 1000
 Max storage for lock list (4KB)              (LOCKLIST) = 128

 Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000
 Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70
 Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128

 Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
 Sort list heap (4KB)                         (SORTHEAP) = 256
 SQL statement heap (4KB)                     (STMTHEAP) = 2048
 Default application heap (4KB)             (APPLHEAPSZ) = 128
```

```
Package cache size (4KB)                 (PCKCACHESZ) = (MAXAPPLS*8)
Statistics heap size (4KB)             (STAT_HEAP_SZ) = 4384

Interval for checking deadlock (ms)         (DLCHKTIME) = 10000
Percent. of lock lists per application       (MAXLOCKS) = 10
Lock timeout (sec)                        (LOCKTIMEOUT) = -1

Changed pages threshold                (CHNGPGS_THRESH) = 60
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 1
Number of I/O servers                   (NUM_IOSERVERS) = 3
Index sort flag                             (INDEXSORT) = YES
Sequential detect flag                      (SEQDETECT) = YES
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC

Track modified pages                         (TRACKMOD) = OFF

Default number of containers                            = 1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32

Max number of active applications            (MAXAPPLS) = AUTOMATIC
Average number of active applications       (AVG_APPLS) = 1
Max DB files open per application            (MAXFILOP) = 64

Log file size (4KB)                         (LOGFILSIZ) = 1000
Number of primary log files                (LOGPRIMARY) = 3
Number of secondary log files               (LOGSECOND) = 2
Changed path to log files                  (NEWLOGPATH) =
Path to log files                                       = /home/db2inst/db2inst
                                                          /NODE0000/SQL00001
                                                          /SQLOGDIR/
Overflow log path                     (OVERFLOWLOGPATH) =
Mirror log path                         (MIRRORLOGPATH) =
First active log file                                   =
Block log on disk full                (BLK_LOG_DSK_FUL) = NO
Percent of max primary log space by transaction(MAX_LOG)= 0
Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0

Group commit count                          (MINCOMMIT) = 1
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100
Log retain for recovery enabled             (LOGRETAIN) = OFF
User exit for logging enabled                (USEREXIT) = OFF

HADR database role                                      = STANDARD
HADR local host name                  (HADR_LOCAL_HOST) =
HADR local service name                (HADR_LOCAL_SVC) =
HADR remote host name                (HADR_REMOTE_HOST) =
HADR remote service name              (HADR_REMOTE_SVC) =
HADR instance name of remote server  (HADR_REMOTE_INST) =
HADR timeout value                       (HADR_TIMEOUT) = 120
HADR log write synchronization mode     (HADR_SYNCMODE) = NEARSYNC

First log archive method                 (LOGARCHMETH1) = OFF
Options for logarchmeth1                  (LOGARCHOPT1) =
Second log archive method                (LOGARCHMETH2) = OFF
Options for logarchmeth2                  (LOGARCHOPT2) =
Failover log archive path                (FAILARCHPATH) =
Number of log archive retries on error   (NUMARCHRETRY) = 5
Log archive retry Delay (secs)         (ARCHRETRYDELAY) = 20
Vendor options                              (VENDOROPT) =

Auto restart enabled                      (AUTORESTART) = ON
Index re-creation time and redo index build  (INDEXREC) = SYSTEM (RESTART)
Log pages during index build            (LOGINDEXBUILD) = OFF
Default number of loadrec sessions    (DFT_LOADREC_SES) = 1
Number of database backups to retain   (NUM_DB_BACKUPS) = 12
Recovery history retention (days)     (REC_HIS_RETENTN) = 366
```

```
TSM management class                    (TSM_MGMTCLASS) =
TSM node name                            (TSM_NODENAME) =
TSM owner                                   (TSM_OWNER) =
TSM password                             (TSM_PASSWORD) =

Automatic maintenance                      (AUTO_MAINT) = OFF
  Automatic database backup            (AUTO_DB_BACKUP) = OFF
  Automatic table maintenance          (AUTO_TBL_MAINT) = OFF
    Automatic runstats                  (AUTO_RUNSTATS) = OFF
    Automatic statistics profiling    (AUTO_STATS_PROF) = OFF
      Automatic profile updates         (AUTO_PROF_UPD) = OFF
    Automatic reorganization               (AUTO_REORG) = OFF
```

The following example shows a portion of the output of the command when you specify the SHOW DETAIL option. The value in the **Delayed Value** column is the value that will be applied the next time you start the instance.

```
                    Database Configuration for Database mick
Description                                Parameter    Current Value  Delayed
                                                                         Value

Database configuration release level                    = 0x0a00
Database release level                                  = 0x0a00

Database territory                                      = en_US
Database code page                                      = 819
Database code set                                       = ISO8859-1
Database country/region code                            = 1
Database collating sequence                             = UNIQUE      UNIQUE
Alternate collating sequence              (ALT_COLLATE) =
Database page size                                      = 4096
Dynamic SQL Query management           (DYN_QUERY_MGMT) = DISABLE     DISABLE
Discovery support for this database       (DISCOVER_DB) = ENABLE      ENABLE
Default query optimization class         (DFT_QUERYOPT) = 5           5
Degree of parallelism                      (DFT_DEGREE) = 1           1
Continue upon arithmetic exceptions   (DFT_SQLMATHWARN) = NO          NO
Default refresh age                   (DFT_REFRESH_AGE) = 0           0
Default maintained table types for opt (DFT_MTTB_TYPES) = SYSTEM      SYSTEM
Number of frequent values retained     (NUM_FREQVALUES) = 10          10
Number of quantiles retained            (NUM_QUANTILES) = 20          20

Backup pending                                          = NO

Database is consistent                                  = YES
Rollforward pending                                     = NO
Restore pending                                         = NO

Multi-page file allocation enabled                      = YES

Log retain for recovery status                          = NO
User exit for logging status                            = NO

Data Links Token Expiry Interval (sec)      (DL_EXPINT) = 60          60
Data Links Write Token Init Expiry Intvl(DL_WT_IEXPINT) = 60          60
Data Links Number of Copies             (DL_NUM_COPIES) = 1           1
Data Links Time after Drop (days)        (DL_TIME_DROP) = 1           1
Data Links Token in Uppercase                (DL_UPPER) = NO          NO
Data Links Token Algorithm                   (DL_TOKEN) = MAC0        MAC0

Database heap (4KB)                            (DBHEAP) = 1200        1200
Size of database shared memory (4KB)  (DATABASE_MEMORY) = AUTOMATIC   AUTOMATIC
                                                          (11516)      (11516)
Catalog cache size (4KB)              (CATALOGCACHE_SZ) = 64           64
Log buffer size (4KB)                        (LOGBUFSZ) = 8            8
Utilities heap size (4KB)                (UTIL_HEAP_SZ) = 5000         5000
Buffer pool size (pages)                     (BUFFPAGE) = 1000         1000
Max storage for lock list (4KB)              (LOCKLIST) = 128          128
Max size of appl. group mem set (4KB) (APPGROUP_MEM_SZ) = 30000        30000
Percent of mem for appl. group heap   (GROUPHEAP_RATIO) = 70           70
Max appl. control heap size (4KB)     (APP_CTL_HEAP_SZ) = 128          128
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES) (SHEAPTHRES)
Sort list heap (4KB)                         (SORTHEAP) = 256          256
SQL statement heap (4KB)                     (STMTHEAP) = 2048         2048
Default application heap (4KB)             (APPLHEAPSZ) = 128          128
Package cache size (4KB)                   (PCKCACHESZ) = (MAXAPPLS*8) (MAXAPPLS*8)
Statistics heap size (4KB)               (STAT_HEAP_SZ) = 4384         4384

Interval for checking deadlock (ms)         (DLCHKTIME) = 10000        10000
Percent. of lock lists per application       (MAXLOCKS) = 10           10
Lock timeout (sec)                        (LOCKTIMEOUT) = -1           -1

Changed pages threshold                (CHNGPGS_THRESH) = 60           60
Number of asynchronous page cleaners   (NUM_IOCLEANERS) = 1            1
Number of I/O servers                   (NUM_IOSERVERS) = 3            3
Index sort flag                             (INDEXSORT) = YES          YES
Sequential detect flag                      (SEQDETECT) = YES          YES
Default prefetch size (pages)         (DFT_PREFETCH_SZ) = AUTOMATIC    AUTOMATIC

Track modified pages                         (TRACKMOD) = NO           NO

Default number of containers                            = 1            1
Default tablespace extentsize (pages)   (DFT_EXTENT_SZ) = 32           32

Max number of active applications            (MAXAPPLS) = AUTOMATIC    AUTOMATIC
                                                           (40)         (40)
Average number of active applications       (AVG_APPLS) = 1            1
Max DB files open per application            (MAXFILOP) = 64           64

Log file size (4KB)                         (LOGFILSIZ) = 1000         1000
Number of primary log files                (LOGPRIMARY) = 3            3
Number of secondary log files               (LOGSECOND) = 2            2
Changed path to log files                  (NEWLOGPATH) =
Path to log files                                       = home/db2inst /home
                                                          /db2inst      /db2inst
                                                          /NODE0000     /db2inst
                                                          /SQL00001     /NODE0000
                                                          /SQLOGDIR/    /SQL00001
                                                                        /SQLOGDIR/
Overflow log path                     (OVERFLOWLOGPATH) =
Mirror log path                         (MIRRORLOGPATH) =
First active log file                                   =
Block log on disk full                (BLK_LOG_DSK_FUL) = NO           NO
Percent of max primary log space by transaction(MAX_LOG)= 0            0
Num. of active log files for 1 active UOW(NUM_LOG_SPAN) = 0            0
Group commit count                          (MINCOMMIT) = 1            1
Percent log file reclaimed before soft chckpt (SOFTMAX) = 100          100
Log retain for recovery enabled             (LOGRETAIN) = OFF          OFF
User exit for logging enabled                (USEREXIT) = OFF          OFF

HADR database role                                      = STANDARD     STANDARD
HADR local host name                  (HADR_LOCAL_HOST) =
HADR local service name                (HADR_LOCAL_SVC) =
HADR remote host name                (HADR_REMOTE_HOST) =
HADR remote service name              (HADR_REMOTE_SVC) =
HADR instance name of remote server  (HADR_REMOTE_INST) =
HADR timeout value                       (HADR_TIMEOUT) = 120          120
HADR log write synchronization mode     (HADR_SYNCMODE) = NEARSYNC     NEARSYNC

First log archive method                 (LOGARCHMETH1) = OFF          OFF
Options for logarchmeth1                  (LOGARCHOPT1) =
Second log archive method                (LOGARCHMETH2) = OFF          OFF
Options for logarchmeth2                  (LOGARCHOPT2) =
Failover log archive path                (FAILARCHPATH) =
Number of log archive retries on error   (NUMARCHRETRY) = 5           5
Log archive retry Delay (secs)         (ARCHRETRYDELAY) = 20          20
Vendor options                              (VENDOROPT) =
Auto restart enabled                      (AUTORESTART) = ON          ON
Index re-creation time and redo index build  (INDEXREC) = SYSTEM      SYSTEM
                                                          (RESTART)   (RESTART)
Log pages during index build            (LOGINDEXBUILD) = OFF         OFF
Default number of loadrec sessions    (DFT_LOADREC_SES) = 1           1
Number of database backups to retain   (NUM_DB_BACKUPS) = 12          12
Recovery history retention (days)     (REC_HIS_RETENTN) = 366         366

TSM management class                    (TSM_MGMTCLASS) =
TSM node name                            (TSM_NODENAME) =
TSM owner                                   (TSM_OWNER) =
TSM password                             (TSM_PASSWORD) =

Automatic maintenance                      (AUTO_MAINT) = OFF         OFF
  Automatic database backup            (AUTO_DB_BACKUP) = OFF         OFF
  Automatic table maintenance          (AUTO_TBL_MAINT) = OFF         OFF
    Automatic runstats                  (AUTO_RUNSTATS) = OFF         OFF
    Automatic statistics profiling    (AUTO_STATS_PROF) = OFF         OFF
    Automatic profile updates           (AUTO_PROF_UPD) = OFF         OFF
    Automatic reorganization               (AUTO_REORG) = OFF          OFF
```

Usage notes:

If an error occurs, the information returned is not valid. If the configuration file is invalid, an error message is returned. The database must be restored from a backup version.

To set the database configuration parameters to the database manager defaults, use the RESET DATABASE CONFIGURATION command.

To retrieve information from all database partitions, use the SYSIBMADM.DBCFG administrative view.

Related tasks:

- "Changing node and database configuration files" in *Administration Guide: Implementation*
- "Configuring DB2 with configuration parameters" on page 1493

Related reference:

- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION " on page 627
- "db2CfgGet - Get the database manager or database configuration parameters" on page 672
- "Configuration parameters summary" on page 1484
- "DBCFG administrative view - Retrieve database configuration parameter information" in *Administrative SQL Routines and Views*

GET DATABASE MANAGER CONFIGURATION

Returns the values of individual entries in the database manager configuration file.

Authorization:

None

Required connection:

None or instance. An instance attachment is not required to perform local DBM configuration operations, but is required to perform remote DBM configuration operations. To display the database manager configuration for a remote instance, it is necessary to first attach to that instance. The `SHOW DETAIL` clause requires an instance attachment.

Command syntax:

```
►►──GET──┬─DATABASE MANAGER─┬──┬─CONFIGURATION─┬──┬─────────────┬──────────►◄
         ├─DB MANAGER───────┤  ├─CONFIG────────┤  └─SHOW DETAIL─┘
         └─DBM──────────────┘  └─CFG───────────┘
```

Command parameters:

SHOW DETAIL
: Displays detailed information showing the current value of database manager configuration parameters as well as the value of the parameters the next time you start the database manager. This option lets you see the result of dynamic changes to configuration parameters.

Examples:

Both node type and platform determine which configuration parameters are listed.

The following is sample output from `GET DATABASE MANAGER CONFIGURATION` (issued on AIX):

```
          Database Manager Configuration

     Node type = Database Server with local clients

 Database manager configuration release level            = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 4.000000e-05

 Max number of concurrently active databases     (NUMDB) = 8
 Data Links support                          (DATALINKS) = NO
 Federated Database System Support           (FEDERATED) = NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) = /usr/java131

 Diagnostic error capture level              (DIAGLEVEL) = 3
 Notify Level                              (NOTIFYLEVEL) = 3
 Diagnostic data directory path               (DIAGPATH) =

 Default database monitor switches
   Buffer pool                         (DFT_MON_BUFPOOL) = OFF
   Lock                                   (DFT_MON_LOCK) = OFF
```

```
  Sort                                  (DFT_MON_SORT) = OFF
  Statement                             (DFT_MON_STMT) = OFF
  Table                                (DFT_MON_TABLE) = OFF
  Timestamp                        (DFT_MON_TIMESTAMP) = ON
  Unit of work                           (DFT_MON_UOW) = OFF
Monitor health of instance and databases   (HEALTH_MON) = ON

SYSADM group name                       (SYSADM_GROUP) =
SYSCTRL group name                     (SYSCTRL_GROUP) =
SYSMAINT group name                   (SYSMAINT_GROUP) =
SYSMON group name                       (SYSMON_GROUP) =

Client Userid-Password Plugin         (CLNT_PW_PLUGIN) =
Client Kerberos Plugin               (CLNT_KRB_PLUGIN) =
Group Plugin                            (GROUP_PLUGIN) =
GSS Plugin for Local Authorization   (LOCAL_GSSPLUGIN) =
Server Plugin Mode                   (SRV_PLUGIN_MODE) = UNFENCED
Server List of GSS Plugins     (SRVCON_GSSPLUGIN_LIST) =
Server Userid-Password Plugin      (SRVCON_PW_PLUGIN) =
Server Connection Authentication         (SRVCON_AUTH) = NOT_SPECIFIED
Database manager authentication       (AUTHENTICATION) = SERVER
Cataloging allowed without authority   (CATALOG_NOAUTH) = YES
Trust all clients                     (TRUST_ALLCLNTS) = YES
Trusted client authentication         (TRUST_CLNTAUTH) = CLIENT
Bypass federated authentication           (FED_NOAUTH) = NO

Default database path                      (DFTDBPATH) = /home/db2inst

Database monitor heap size (4KB)         (MON_HEAP_SZ) = 90
Java Virtual Machine heap size (4KB)    (JAVA_HEAP_SZ) = 512
Audit buffer size (4KB)                 (AUDIT_BUF_SZ) = 0
Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC
Backup buffer default size (4KB)           (BACKBUFSZ) = 1024
Restore buffer default size (4KB)          (RESTBUFSZ) = 1024

Sort heap threshold (4KB)                 (SHEAPTHRES) = 20000

Directory cache support                    (DIR_CACHE) = YES

Application support layer heap size (4KB)   (ASLHEAPSZ) = 15
Max requester I/O block size (bytes)         (RQRIOBLK) = 32767
Query heap size (4KB)                   (QUERY_HEAP_SZ) = 1000

Workload impact by throttled utilities(UTIL_IMPACT_LIM) = 10

Priority of agents                           (AGENTPRI) = SYSTEM
Max number of existing agents               (MAXAGENTS) = 200
Agent pool size                        (NUM_POOLAGENTS) = 100(calculated)
Initial number of agents in pool       (NUM_INITAGENTS) = 0
Max number of coordinating agents     (MAX_COORDAGENTS) = MAXAGENTS
Max no. of concurrent coordinating agents  (MAXCAGENTS) = MAX_COORDAGENTS
Max number of client connections      (MAX_CONNECTIONS) = MAX_COORDAGENTS

Keep fenced process                        (KEEPFENCED) = YES
Number of pooled fenced processes         (FENCED_POOL) = MAX_COORDAGENTS
Initial number of fenced processes     (NUM_INITFENCED) = 0

Index re-creation time and redo index build  (INDEXREC) = RESTART

Transaction manager database name         (TM_DATABASE) = 1ST_CONN
Transaction resync interval (sec)     (RESYNC_INTERVAL) = 180

SPM name                                     (SPM_NAME) =
SPM log size                          (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit                 (SPM_MAX_RESYNC) = 20
SPM log path                             (SPM_LOG_PATH) =
TCP/IP Service name                          (SVCENAME) =
Discovery mode                               (DISCOVER) = SEARCH
Discover server instance                (DISCOVER_INST) = ENABLE

Maximum query degree of parallelism   (MAX_QUERYDEGREE) = ANY
Enable intra-partition parallelism     (INTRA_PARALLEL) = NO

No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC
No. of int. communication channels (FCM_NUM_CHANNELS)   = AUTOMATIC
```

The following output sample shows the information displayed when you specify the `WITH DETAIL` option. The value that appears in the **Delayed Value** is the value that will be in effect the next time you start the database manager instance.

```
                        Database Manager Configuration
 Node type = Database Server with local clients
 Description                                  Parameter   Current Value  Delayed
                                                                          Value

Database manager configuration release level           = 0x0a00

 CPU speed (millisec/instruction)             (CPUSPEED) = 4.000000e     4.000000e
                                                            -05           -05
 Max number of concurrently active databases     (NUMDB) = 8             8
 Data Links support                          (DATALINKS) = NO            NO
 Federated Database System Support           (FEDERATED) = NO            NO
 Transaction processor monitor name        (TP_MON_NAME) =

 Default charge-back account           (DFT_ACCOUNT_STR) =

 Java Development Kit installation path       (JDK_PATH) = /wsdb/v81     /usr
                                                            /bldsupp      /java131
                                                            /AIX/jdk1.3.1

 Diagnostic error capture level              (DIAGLEVEL) = 3            3
 Notify Level                              (NOTIFYLEVEL) = 3            3
 Diagnostic data directory path               (DIAGPATH) =

 Default database monitor switches
   Buffer pool                         (DFT_MON_BUFPOOL) = OFF         OFF
   Lock                                   (DFT_MON_LOCK) = OFF         OFF
   Sort                                   (DFT_MON_SORT) = OFF         OFF
   Statement                              (DFT_MON_STMT) = OFF         OFF
   Table                                 (DFT_MON_TABLE) = OFF         OFF
   Timestamp                         (DFT_MON_TIMESTAMP) = ON          ON
   Unit of work                            (DFT_MON_UOW) = OFF         OFF
 Monitor health of instance and databases   (HEALTH_MON) = ON          ON

 SYSADM group name                        (SYSADM_GROUP) = BUILD
 SYSCTRL group name                      (SYSCTRL_GROUP) =
 SYSMAINT group name                    (SYSMAINT_GROUP) =
 SYSMON group name                        (SYSMON_GROUP) =

 Client Userid-Password Plugin          (CLNT_PW_PLUGIN) =

 Client Kerberos Plugin                (CLNT_KRB_PLUGIN) =

 Group Plugin                             (GROUP_PLUGIN) =

 GSS Plugin for Local Authorization    (LOCAL_GSSPLUGIN) =

 Server Plugin Mode                    (SRV_PLUGIN_MODE) = UNFENCED    UNFENCED

 Server List of GSS Plugins      (SRVCON_GSSPLUGIN_LIST) =

 Server Userid-Password Plugin        (SRVCON_PW_PLUGIN) =
 Server Connection Authentication         (SRVCON_AUTH) = NOT_          NOT_
                                                           SPECIFIED    SPECIFIED
 Database manager authentication       (AUTHENTICATION) = SERVER      SERVER
 Cataloging allowed without authority   (CATALOG_NOAUTH) = YES         YES
 Trust all clients                      (TRUST_ALLCLNTS) = YES         YES
 Trusted client authentication          (TRUST_CLNTAUTH) = CLIENT      CLIENT
 Bypass federated authentication            (FED_NOAUTH) = NO          NO

 Default database path                       (DFTDBPATH) = /home       /home
                                                            /db2inst    /db2inst

 Database monitor heap size (4KB)          (MON_HEAP_SZ) = 90          90
 Java Virtual Machine heap size (4KB)     (JAVA_HEAP_SZ) = 512         512
 Audit buffer size (4KB)                  (AUDIT_BUF_SZ) = 0           0
 Size of instance shared memory (4KB)  (INSTANCE_MEMORY) = AUTOMATIC   AUTOMATIC
                                                            (5386)       (20)
 Backup buffer default size (4KB)            (BACKBUFSZ) = 1024        1024
 Restore buffer default size (4KB)           (RESTBUFSZ) = 1024        1024

 Sort heap threshold (4KB)                  (SHEAPTHRES) = 20000       20000

 Directory cache support                     (DIR_CACHE) = YES         YES

 Application support layer heap size (4KB)   (ASLHEAPSZ) = 15          15
 Max requester I/O block size (bytes)         (RQRIOBLK) = 32767       32767
 Query heap size (4KB)                   (QUERY_HEAP_SZ) = 1000        1000

 Workload impact by throttled utilities(UTIL_IMPACT_LIM) = 10          10

 Priority of agents                           (AGENTPRI) = SYSTEM      SYSTEM
 Max number of existing agents               (MAXAGENTS) = 200         200
 Agent pool size                        (NUM_POOLAGENTS) = 100         100
                                                                       (calculated)
 Initial number of agents in pool       (NUM_INITAGENTS) = 0           0
 Max number of coordinating agents     (MAX_COORDAGENTS) = 200         MAXAGENTS
 Max no. of concurrent coordinating agents  (MAXCAGENTS) = 200         MAX_COORDAGENTS
 Max number of client connections      (MAX_CONNECTIONS) = 200         MAX_COORDAGENTS

 Keep fenced process                        (KEEPFENCED) = YES         YES
 Number of pooled fenced processes         (FENCED_POOL) = MAX_        MAX_
                                                           COORDAGENTS  COORDAGENTS
 Initial number of fenced processes     (NUM_INITFENCED) = 0           0

 Index re-creation time and redo index build  (INDEXREC) = RESTART     RESTART

 Transaction manager database name         (TM_DATABASE) = 1ST_CONN    1ST_CONN
 Transaction resync interval (sec)     (RESYNC_INTERVAL) = 180         180

 SPM name                                     (SPM_NAME) =
 SPM log size                          (SPM_LOG_FILE_SZ) = 256         256
 SPM resync agent limit                 (SPM_MAX_RESYNC) = 20          20
 SPM log path                             (SPM_LOG_PATH) =

 TCP/IP Service name                          (SVCENAME) =
 Discovery mode                               (DISCOVER) = SEARCH      SEARCH
 Discover server instance                (DISCOVER_INST) = ENABLE      ENABLE

 Maximum query degree of parallelism   (MAX_QUERYDEGREE) = ANY         ANY
 Enable intra-partition parallelism     (INTRA_PARALLEL) = NO          NO

 No. of int. communication buffers(4KB)(FCM_NUM_BUFFERS) = AUTOMATIC   AUTOMATIC
 No. of int. communication channels (FCM_NUM_CHANNELS)   = AUTOMATIC   AUTOMATIC
```

Usage notes:

- If an attachment to a remote instance or a different local instance exists, the database manager configuration parameters for the attached server are returned; otherwise, the local database manager configuration parameters are returned.
- If an error occurs, the information returned is invalid. If the configuration file is invalid, an error message is returned. The user must install the database manager again to recover.
- To set the configuration parameters to the default values shipped with the database manager, use the `RESET DATABASE MANAGER CONFIGURATION` command.
- The `AUTOMATIC` values indicated on `get database manager configuration show detail` for `FCM_NUM_BUFFERS` and `FCM_NUM_CHANNELS` are the initial values at instance startup time and do not reflect any automatic increasing/decreasing that might have occurred during runtime.

Related tasks:
- "Changing node and database configuration files" in *Administration Guide: Implementation*
- "Configuring DB2 with configuration parameters" on page 1493

Related reference:
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629
- "db2CfgGet - Get the database manager or database configuration parameters" on page 672
- "Configuration parameters summary" on page 1484
- "DBMCFG administrative view - Retrieve database manager configuration parameter information" in *Administrative SQL Routines and Views*

IMPORT

Inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. LOAD is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Authorization:
- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*

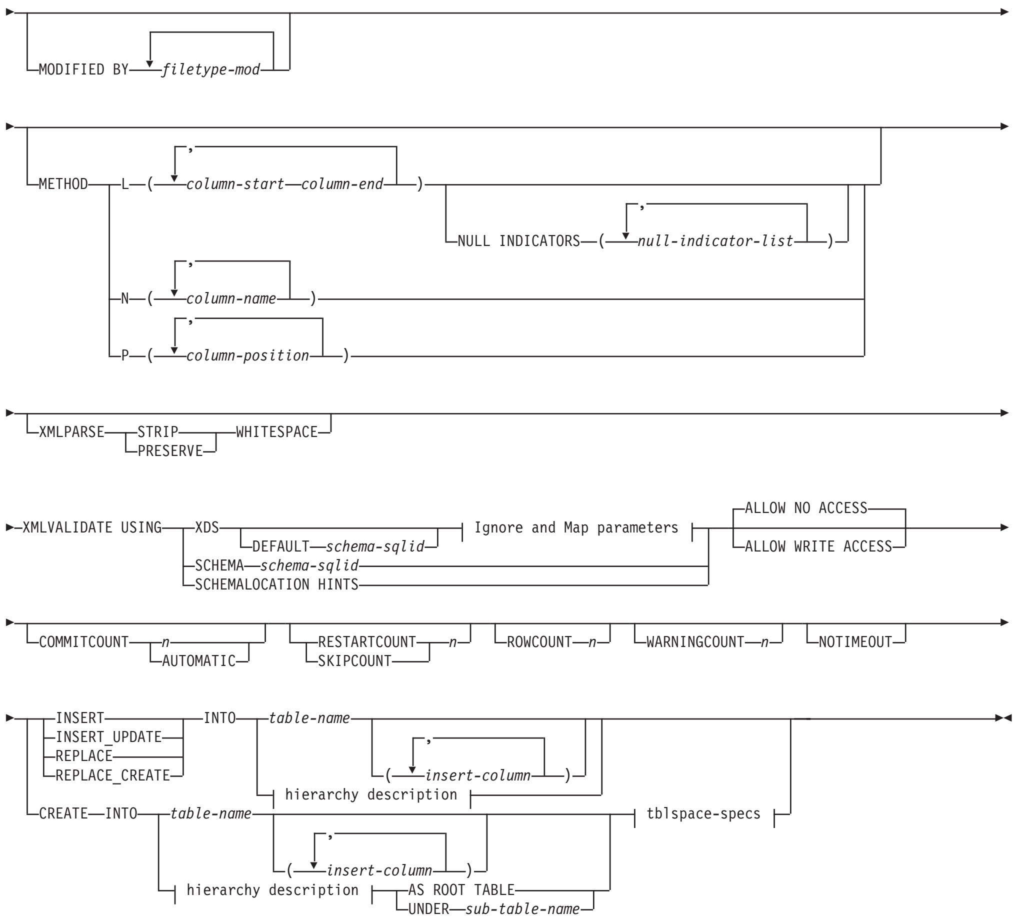
  - *dbadm*
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database and USE privilege on the table space, as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATIN privilege on the schema, if the schema name of the table refers to an existing schema
- IMPORT to a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database and USE privilege on the table space and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on every sub-table in the hierarchy
- To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the import fails and an error (SQLSTATE 42512) is returned.
- To import data into a table that has protected rows, the session authorization ID must hold LBAC credentials that meets these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access

  The label on the row to insert, the user's LBAC credentials, the security policy definition, and the LBAC rules determine determine the label on the row.
- If the REPLACE or REPLACE_CREATE option is specified, the session authorization ID must have the authority to drop the table.

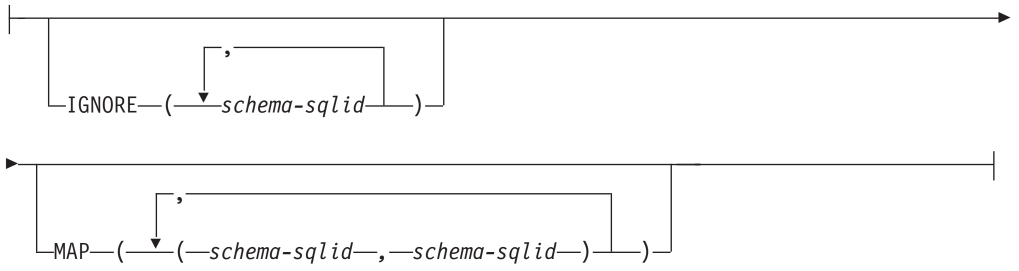
Required connection:

Command syntax:

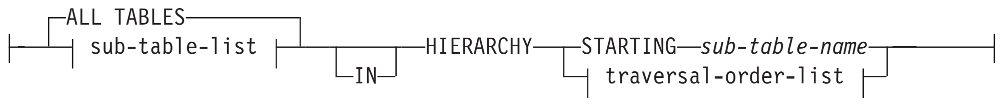
```
>>--IMPORT FROM--filename--OF--filetype--+-----------------------+--+-----------------------+---------->
                                         |           .-,--------. |  |          .-,--------. |
                                         |           V          | |  |          V          | |
                                         '-LOBS FROM----lob-path-+-'  '-XML FROM----xml-path-+-'
```



Ignore and Map parameters:



hierarchy description:



sub-table-list:

```
         ┌─,──────────────────────────────────────────┐
         ▼                                            │
├──(─────sub-table-name──┬──────────────────────────┬─┴──)──────────────────────┤
                         │     ┌─,────────────┐     │
                         │     ▼              │     │
                         └─(───insert-column──┴──)──┘
```

traversal-order-list:

```
         ┌─,──────────────┐
         ▼                │
├──(─────sub-table-name───┴──)──────────────────────────────────────────────────┤
```

tblspace-specs:

```
├──┬──────────────────────────────────────────────────────────────────────────┬──┤
   └─IN──tablespace-name──┬─────────────────────────────┬──┬──────────────────────────┬─┘
                          └─INDEX IN──tablespace-name───┘  └─LONG IN──tablespace-name─┘
```

Command parameters:

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal order.

ALLOW NO ACCESS
Runs import in the offline mode. An exclusive (X) lock on the target table is acquired before any rows are inserted. This prevents concurrent applications from accessing table data. This is the default import behavior.

ALLOW WRITE ACCESS
Runs import in the online mode. An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the REPLACE, CREATE, or REPLACE_CREATE import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the COMMITCOUNT option was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and COMMITCOUNT must be specified with a valid number (AUTOMATIC is not considered a valid option).

AS ROOT TABLE
Creates one or more sub-tables as a stand-alone table hierarchy.

COMMITCOUNT *n*/AUTOMATIC
Performs a COMMIT after every *n* records are imported. When a number *n* is specified, import performs a COMMIT after every *n* records are imported. When compound inserts are used, a user-specified commit frequency of *n* is rounded up to the first integer multiple of the compound count value. When AUTOMATIC is specified, import internally determines when a commit needs to be performed. The utility will commit for either one of two reasons:

- to avoid running out of active log space
- to avoid lock escalation from row level to table level

If the ALLOW WRITE ACCESS option is specified, and the COMMITCOUNT option is not specified, the import utility will perform commits as if COMMITCOUNT AUTOMATIC had been specified.

If the `IMPORT` command encounters an SQL0964C (Transaction Log Full) while inserting or updating a record and the `COMMITCOUNT` *n* is specified, `IMPORT` will attempt to resolve the issue by performing an unconditional commit and then reattempt to insert or update the record. If this does not help resolve the log full condition (which would be the case when the log full is attributed to other activity on the database), then the `IMPORT` command will fail as expected, however the number of rows committed may not be a multiple of the `COMMITCOUNT` *n* value. The `RESTARTCOUNT` or `SKIPCOUNT` option can be used to avoid processing those row already committed.

CREATE

Creates the table definition and row contents in the code page of the database. If the data was exported from a DB2 table, sub-table, or hierarchy, indexes are created. If this option operates on a hierarchy, and data was exported from DB2, a type hierarchy will also be created. This option can only be used with IXF files.

This parameter is not valid when you import to a nickname.

Note: If the data was exported from an MVS host database, and it contains LONGVAR fields whose lengths, calculated on the page size, are less than 254, CREATE might fail because the rows are too long. See Using import to recreate an exported table for a list of restrictions. In this case, the table should be created manually, and IMPORT with INSERT should be invoked, or, alternatively, the LOAD command should be used.

DEFAULT schema-sqlid

This option can only be used when the USING XDS parameter is specified. The schema specified through the DEFAULT clause identifies a schema to use for validation when the XML Data Specifier (XDS) of an imported XML document does not contain an SCH attribute identifying an XML Schema.

The DEFAULT clause takes precedence over the IGNORE and MAP clauses. If an XDS satisfies the DEFAULT clause, the IGNORE and MAP specifications will be ignored.

FROM filename

HIERARCHY

Specifies that hierarchical data is to be imported.

IGNORE schema-sqlid

This option can only be used when the USING XDS parameter is specified. The IGNORE clause specifies a list of one or more schemas to ignore if they are identified by an SCH attribute. If an SCH attribute exists in the XML Data Specifier for an imported XML document, and the schema identified by the SCH attribute is included in the list of schemas to IGNORE, then no schema validation will occur for the imported XML document.

If a schema is specified in the IGNORE clause, it cannot also be present in the left side of a schema pair in the MAP clause.

The IGNORE clause applies only to the XDS. A schema that is mapped by the MAP clause will not be subsequently ignored if specified by the IGNORE clause.

IN tablespace-name
Identifies the table space in which the table will be created. The table space must exist, and must be a REGULAR table space. If no other table space is specified, all table parts are stored in this table space. If this clause is not specified, the table is created in a table space created by the authorization ID. If none is found, the table is placed into the default table space USERSPACE1. If USERSPACE1 has been dropped, table creation fails.

INDEX IN tablespace-name
Identifies the table space in which any indexes on the table will be created. This option is allowed only when the primary table space specified in the IN clause is a DMS table space. The specified table space must exist, and must be a REGULAR or LARGE DMS table space.

Note: Specifying which table space will contain an index can only be done when the table is created.

insert-column
Specifies the name of a column in the table or the view into which data is to be inserted.

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds rows of imported data to the target table, or updates existing rows (of the target table) with matching primary keys.

INTO table-name
Specifies the database table into which the data is to be imported. This table cannot be a system table, a declared temporary table or a summary table.

One can use an alias for INSERT, INSERT_UPDATE, or REPLACE, except in the case of a down-level server, when the fully qualified or the unqualified table name should be used. A qualified table name is in the form: *schema.tablename*. The *schema* is the user name under which the table was created.

LOBS FROM lob-path
The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behaviour.

This parameter is not valid when you import to a nickname.

LONG IN tablespace-name
Identifies the table space in which the values of any long columns (LONG VARCHAR, LONG VARGRAPHIC, LOB data types, or distinct types with any of these as source types) will be stored. This option is allowed only if the primary table space specified in the IN clause is a DMS table space. The table space must exist, and must be a LARGE DMS table space.

MAP schema-sqlid
This option can only be used when the USING XDS parameter is specified.

Use the MAP clause to specify alternate schemas to use in place of those specified by the SCH attribute of an XML Data Specifier (XDS) for each imported XML document. The MAP clause specifies a list of one or more schema pairs, where each pair represents a mapping of one schema to another. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

If a schema is present in the left side of a schema pair in the MAP clause, it cannot also be specified in the IGNORE clause.

Once a schema pair mapping is applied, the result is final. The mapping operation is non-transitive, and therefore the schema chosen will not be subsequently applied to another schema pair mapping.

A schema cannot be mapped more than once, meaning that it cannot appear on the left side of more than one pair.

METHOD

L Specifies the start and end column numbers from which to import data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1.

Note: This method can only be used with ASC files, and is the only valid option for that file type.

N Specifies the names of the columns to be imported.

Note: This method can only be used with IXF files.

P Specifies the field numbers of the input data fields to be imported.

Note: This method can only be used with IXF or DEL files, and is the only valid option for the DEL file type.

MODIFIED BY filetype-mod
Specifies file type modifier options. See File type modifiers for the import utility.

NOTIMEOUT
Specifies that the import utility will not time out while waiting for locks. This option supersedes the *locktimeout* database configuration parameter. Other applications are not affected.

NULL INDICATORS null-indicator-list
This option can only be used when the METHOD L parameter is specified. That is, the input file is an ASC file. The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the METHOD L parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the METHOD L option will be imported.

The NULL indicator character can be changed using the MODIFIED BY option, with the `nullindchar` file type modifier.

OF filetype
Specifies the format of the data in the input file:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format), which is used by a variety of database manager and file manager programs
- WSF (work sheet format), which is used by programs such as:
  - Lotus 1-2-3
  - Lotus Symphony
- IXF (integrated exchange format, PC version), which means it was exported from the same or another DB2 table. An IXF file also contains the table definition and definitions of any existing indexes, except when columns are specified in the SELECT statement.

Th WSF file type is not supported when you import to a nickname.

REPLACE
Deletes all existing data from the table by truncating the data object, and inserts the imported data. The table definition and the index definitions are not changed. This option can only be used if the table exists. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

This option does not honour the CREATE TABLE statement's NOT LOGGED INITIALLY (NLI) clause or the ALTER TABLE statement's ACTIVE NOT LOGGED INITIALLY clause.

If an import with the REPLACE option is performed within the same transaction as a CREATE TABLE or ALTER TABLE statement where the NLI clause is invoked, the import will not honor the NLI clause. All inserts will be logged.

Workaround 1
Delete the contents of the table using the DELETE statement, then invoke the import with INSERT statement

Workaround 2
Drop the table and recreate it, then invoke the import with INSERT statement.

This limitation applies to DB2 UDB Version 7 and DB2 UDB Version 8

REPLACE_CREATE
If the table exists, deletes all existing data from the table by truncating the data object, and inserts the imported data without changing the table definition or the index definitions.

If the table does not exist, creates the table and index definitions, as well as the row contents, in the code page of the database. See Using import to recreate an exported table for a list of restrictions.

This option can only be used with IXF files. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

This parameter is not valid when you import to a nickname.

RESTARTCOUNT *n*
Specifies that an import operation is to be started at record *n* + 1. The first

n records are skipped. This option is functionally equivalent to SKIPCOUNT. RESTARTCOUNT and SKIPCOUNT are mutually exclusive.

ROWCOUNT *n*
: Specifies the number *n* of physical records in the file to be imported (inserted or updated). Allows a user to import only *n* rows from a file, starting from the record determined by the SKIPCOUNT or RESTARTCOUNT options. If the SKIPCOUNT or RESTARTCOUNT options are not specified, the first *n* rows are imported. If SKIPCOUNT *m* or RESTARTCOUNT *m* is specified, rows *m*+1 to *m*+*n* are imported. When compound inserts are used, user specified rowcount n is rounded up to the first integer multiple of the compound count value.

SKIPCOUNT *n*
: Specifies that an import operation is to be started at record *n* + 1. The first *n* records are skipped. This option is functionally equivalent to RESTARTCOUNT. SKIPCOUNT and RESTARTCOUNT are mutually exclusive.

STARTING sub-table-name
: A keyword for hierarchy only, requesting the default order, starting from *sub-table-name*. For PC/IXF files, the default order is the order stored in the input file. The default order is the only valid order for the PC/IXF file format.

sub-table-list
: For typed tables with the INSERT or the INSERT_UPDATE option, a list of sub-table names is used to indicate the sub-tables into which data is to be imported.

traversal-order-list
: For typed tables with the INSERT, INSERT_UPDATE, or the REPLACE option, a list of sub-table names is used to indicate the traversal order of the importing sub-tables in the hierarchy.

UNDER sub-table-name
: Specifies a parent table for creating one or more sub-tables.

WARNINGCOUNT *n*
: Stops the import operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail. If *n* is zero, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

XML FROM xml-path
: Specifies one or more paths that contain the XML files.

XMLPARSE
: Specifies how XML documents are parsed. If this option is not specified, the parsing behaviour for XML documents will be determined by the value of the CURRENT XMLPARSE OPTION special register.

  STRIP WHITESPACE
  : Specifies to remove whitespace when the XML document is parsed.

  PRESERVE WHITESPACE
  : Specifies not to remove whitespace when the XML document is parsed.

XMLVALIDATE
Specifies that XML documents are validated against a schema, when applicable.

USING XDS
XML documents are validated against the XML schema identified by the XML Data Specifier (XDS) in the main data file. By default, if the XMLVALIDATE option is invoked with the USING XDS clause, the schema used to perform validation will be determined by the SCH attribute of the XDS. If an SCH attribute is not present in the XDS, no schema validation will occur unless a default schema is specified by the DEFAULT clause.

The DEFAULT, IGNORE, and MAP clauses can be used to modify the schema determination behavior. These three optional clauses apply directly to the specifications of the XDS, and not to each other. For example, if a schema is selected because it is specified by the DEFAULT clause, it will not be ignored if also specified by the IGNORE clause. Similarly, if a schema is selected because it is specified as the first part of a pair in the MAP clause, it will not be re-mapped if also specified in the second part of another MAP clause pair.

USING SCHEMA schema-sqlid
XML documents are validated against the XML schema with the specified SQL identifier. In this case, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

USING SCHEMALOCATION HINTS
XML documents are validated against the schemas identified by XML schema location hints in the source XML documents. If a schemaLocation (SCH) attribute is not found in the XML document, no validation will occur. When the USING SCHEMALOCATION HINTS clause is specified, the SCH attribute of the XML Data Specifier (XDS) will be ignored for all XML columns.

See examples of the XMLVALIDATE option below.

Usage notes:

Be sure to complete all table operations and release all locks before starting an import operation. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

The import utility adds rows to the target table using the SQL INSERT statement. The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:

- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the RESTARTCOUNT parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the COMMITCOUNT parameter is not zero. If automatic COMMITs are not performed, a full log results in a ROLLBACK.

Offline import does not perform automatic COMMITs if any of the following conditions is true:
- the target is a view, not a table
- compound inserts are used
- buffered inserts are used

By default, online import performs automatic COMMITs to free both the active log space and the lock list. Automatic COMMITs are not performed only if a COMMITCOUNT value of zero is specified.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

One cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:
1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions *are* preserved if the data was previously exported using `SELECT *`.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60KB), the message file returned to the user on the client might be missing messages from the middle of the import operation. The first 30KB of message information and the last 30KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes.

The database table or hierarchy must exist before data in the ASC, DEL, or WSF file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand. The file copying step is not necessary if the source and the target databases are both accessible from the same client.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 clients on the AIX operating system.

For table objects on an 8 KB page that are close to the limit of 1012 columns, import of PC/IXF data files might cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of DEL or ASC files. If PC/IXF files are being used to create a new table, an alternative is use `db2look` to dump the DDL statement that created the table, and then to issue that statement through the CLP.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The RESTARTCOUNT parameter, but not the COMMITCOUNT parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file.

The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

On the Windows operating system:
- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Security labels in their internal format might contain newline characters. If you import the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the delprioritychar file type modifier in the IMPORT command.

Federated considerations:

When using the IMPORT command and the INSERT, UPDATE, or INSERT_UPDATE command parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you wish to use when doing an import operation already exists. There are also several restrictions you should be aware of as shown in the IMPORT command parameters section.

Related concepts:
- "Import Overview" in *Data Movement Utilities Guide and Reference*
- "Privileges, authorities, and authorization required to use import" in *Data Movement Utilities Guide and Reference*

Related tasks:
- "Importing data" in *Data Movement Utilities Guide and Reference*

Related reference:
- "XMLPARSE scalar function" in *SQL Reference, Volume 1*
- "ADMIN_CMD procedure ÔÇô Run administrative commands" in *Administrative SQL Routines and Views*
- "db2look - DB2 statistics and DDL extraction tool command" in *Command Reference*
- "IMPORT command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "Import sessions - CLP examples" in *Data Movement Utilities Guide and Reference*
- "LOB and XML file behavior with regard to import and export" in *Data Movement Utilities Guide and Reference*

INSPECT

Inspect database for architectural integrity, checking the pages of the database for page consistency. The inspection checks the structures of table objects and structures of table spaces are valid.

Scope:

In a single partition database environment, the scope is that single partition only. In a partitioned database system, it is the collection of all logical partitions defined in db2nodes.cfg. For partitioned tables, the CHECK DATABASE and CHECK TABLESPACE options include individual data partitions and non-partitioned indexes. The CHECK TABLE option is also available for a partitioned table, however it will check all data partitions and indexes in a table, rather than checking a single data partition or index.

Authorization:

For INSPECT CHECK, one of the following:
- *sysadm*
- *dbadm*
- *sysctrl*
- *sysmaint*
- CONTROL privilege if single table.

Required Connection:

Database

Command Syntax:

```
>>--INSPECT--+-| Check Clause |-----------------------+--+----------------------+------>
             '-| Row Compression Estimate Clause |-'  '-FOR ERROR STATE ALL-'

   .-LIMIT ERROR TO DEFAULT---.
>--+--------------------------+--+------------------+--RESULTS--+------+-------------->
   '-LIMIT ERROR TO--+-n---+--'  '-| Level Clause |-'           '-KEEP-'
                     '-ALL-'

>--filename--+---------------------------------+------------------------------------><
             '-| On Database Partition Clause |-'
```

Check Clause:

```
|--CHECK--+--+-DATABASE--+----------------------------------+--------------+--+--|
          |  |           '-BEGIN TBSPACEID--n--+-------------+-'              |  |
          |  |                                  '-OBJECTID--n-'               |  |
          |  '-TABLESPACE--+-NAME--tablespace-name-+--+--------------------+--'  |
          |                '-TBSPACEID--n----------'  '-BEGIN OBJECTID--n-'      |
          '-TABLE--+-NAME--table-name--+----------------------+-+-----------------'
                   |                   '-SCHEMA--schema-name-' |
                   '-TBSPACEID--n--OBJECTID--n-----------------'
```

Row Compression Estimate Clause:

```
├──ROWCOMPESTIMATE-TABLE──┬──NAME──table-name──┬──────────────────────┬─┬──────────┤
                          │                    └──SCHEMA──schema-name──┘ │
                          └──TBSPACEID──n──OBJECTID──n───────────────────┘
```

Level Clause:

```
    ┌─EXTENTMAP NORMAL─────────┐  ┌─DATA NORMAL──────────┐  ┌─BLOCKMAP NORMAL─────────┐
├───┼──────────────────────────┼──┼──────────────────────┼──┼─────────────────────────┼──────────►
    └─EXTENTMAP───┬─NONE─┬─────┘  └─DATA───┬─NONE─┬─────┘  └─BLOCKMAP───┬─NONE─┬─────┘
                  └─LOW──┘                 └─LOW──┘                     └─LOW──┘

    ┌─INDEX NORMAL─────────┐  ┌─LONG NORMAL──────────┐  ┌─LOB NORMAL──────────┐
►───┼──────────────────────┼──┼──────────────────────┼──┼─────────────────────┼──────────────────►
    └─INDEX───┬─NONE─┬─────┘  └─LONG───┬─NONE─┬─────┘  └─LOB───┬─NONE─┬─────┘
              └─LOW──┘                 └─LOW──┘               └─LOW──┘

    ┌─XML NORMAL─────────┐
►───┼────────────────────┼───────────────────────────────────────────────────────────────────┤
    └─XML───┬─NONE─┬─────┘
            └─LOW──┘
```

On Database Partition Clause:

```
├──ON──┬─┤ Database Partition List Clause ├────────────────────────────────┬──┤
       └─ALL DBPARTITIONNUMS──┬───────────────────────────────────────────┬─┘
                              └─EXCEPT─┤ Database Partition List Clause ├─┘
```

Database Partition List Clause:

```
├──┬─DBPARTITIONNUM──┬──────────────────────────────────────────────────────────►
   └─DBPARTITIONNUMS─┘


     ┌─,────────────────────────────────────────────────────┐
     ▼                                                      │
►─(──db-partition-number1──┬───────────────────────────────┬┴──)─────────────┤
                           └─TO──db-partition-number2──────┘
```

Command Parameters:

CHECK
Specifies check processing.

DATABASE
Specifies whole database.

BEGIN TBSPACEID n
Specifies processing to begin from table space with given table space ID number.

BEGIN TBSPACEID n OBJECTID n
Specifies processing to begin from table with given table space ID number and object ID number.

TABLESPACE

NAME tablespace-name
Specifies single table space with given table space name.

TBSPACEID n
Specifies single table space with given table space ID number.

BEGIN OBJECTID n
Specifies processing to begin from table with given object ID number.

TABLE

NAME table-name
Specifies table with given table name.

SCHEMA schema-name
Specifies schema name for specified table name for single table operation.

TBSPACEID n OBJECTID n
Specifies table with given table space ID number and object ID number.

ROWCOMPESTIMATE
Estimates the effectiveness of row compression for a table. You can also specify which database partition(s) this operation is to be done on.

This tool is capable of taking a sample of the table data, and building a dictionary from it. This dictionary can then be used to test compression against the records contained in the sample. From this test compression, data is be gathered from which the following estimates are made:

- Percentage of bytes saved from compression
- Percentage of pages saved from compression
- Percentage rows ineligible for compression due to small data size
- Compression dictionary size
- Expansion dictionary size

`INSPECT` will insert the dictionary built for gathering these compression estimates if the `COMPRESS YES` attribute is set for this table, and a dictionary does not already exist for this table. `INSPECT` will attempt to insert the dictionary concurrent to other applications accessing the table. Dictionary insert requires an Exclusive Table Alter lock and an Intent on Exclusive Table lock. `INSPECT` will only insert a dictionary into tables that support Row Compression. For partitioned tables, a separate dictionary is built and inserted on each partition.

RESULTS
Specifies the result output file. The file will be written out to the diagnostic data directory path. If there is no error found by the check processing, this result output file will be erased at the end of the INSPECT operation. If there are errors found by the check processing, this result output file will not be erased at the end of the INSPECT operation.

KEEP Specifies to always keep the result output file.

file-name
Specifies the name for the result output file.

ALL DBPARTITIONNUMS
Specifies that operation is to be done on all database partitions specified in the db2nodes.cfg file. This is the default if a node clause is not specified.

EXCEPT
Specifies that operation is to be done on all database partitions specified in the db2nodes.cfg file, except those specified in the node list.

ON DBPARTITIONNUM / ON DBPARTITIONNUMS
Perform operation on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from db-partition-number1 up to and including db-partition-number2 are included in the database partition list.

FOR ERROR STATE ALL
For table object with internal state already indicating error state, the check will just report this status and not scan through the object. Specifying this option will have the processing scan through the object even if internal state already lists error state.

LIMIT ERROR TO *n*
Number of pages in error for an object to limit reporting for. When this limit of the number of pages in error for an object is reached, the processing will discontinue the check on the rest of the object.

LIMIT ERROR TO DEFAULT
Default number of pages in error for an object to limit reporting for. This value is the extent size of the object. This parameter is the default.

LIMIT ERROR TO ALL
No limit on number of pages in error reported.

EXTENTMAP

NORMAL
Specifies processing level is normal for extent map. Default.

NONE
Specifies processing level is none for extent map.

LOW Specifies processing level is low for extent map.

DATA

NORMAL
Specifies processing level is normal for data object. Default.

NONE
Specifies processing level is none for data object.

LOW Specifies processing level is low for data object.

BLOCKMAP

NORMAL
Specifies processing level is normal for block map object. Default.

NONE
Specifies processing level is none for block map object.

LOW Specifies processing level is low for block map object.

INDEX

NORMAL
Specifies processing level is normal for index object. Default.

NONE
Specifies processing level is none for index object.

LOW Specifies processing level is low for index object.

LONG

NORMAL
Specifies processing level is normal for long object. Default.

NONE
Specifies processing level is none for long object.

LOW Specifies processing level is low for long object.

LOB

NORMAL
Specifies processing level is normal for LOB object. Default.

NONE
Specifies processing level is none for LOB object.

LOW Specifies processing level is low for LOB object.

XML

NORMAL
Specifies processing level is normal for XML column object. Default. Pages of XML object will be checked for most inconsistencies. Actual XML data will not be inspected.

NONE
Specifies processing level is none for XML column object. XML object will not be inspected at all.

LOW Specifies processing level is low for XML column object. Pages of XML object will be checked for some inconsistencies. Actual XML data will not be inspected.

Usage Notes:

1. For check operations on table objects, the level of processing can be specified for the objects. The default is NORMAL level, specifying NONE for an object excludes it. Specifying LOW will do subset of checks that are done for NORMAL.
2. The check database can be specified to start from a specific table space or from a specific table by specifying the ID value to identify the table space or the table.
3. The check table space can be specified to start from a specific table by specifying the ID value to identify the table.
4. The processing of table spaces will affect only the objects that reside in the table space.
5. The online inspect processing will access database objects using isolation level uncommitted read. COMMIT processing will be done during INSPECT processing. It is advisable to end the unit of work by issuing a COMMIT or ROLLBACK before invoking INSPECT.
6. The online inspect check processing will write out unformatted inspection data results to the results file specified. The file will be written out to the diagnostic

data directory path. If there is no error found by the check processing, this result output file will be erased at the end of INSPECT operation. If there are errors found by the check processing, this result output file will not be erased at the end of INSPECT operation. After check processing completes, to see inspection details, the inspection result data will require to be formatted out with the utility db2inspf. The results file will have file extension of the database partition number.

7. In a partitioned database environment, each database partition will generate its own results output file with extension corresponding to its database partition number. The output location for the results output file will be the database manager diagnostic data directory path. If the name of a file that already exists is specified, the operation will not be processed, the file will have to be removed before that file name can be specified.
8. Normal online inspect processing will access database objects using isolation level uncommitted read. Inserting a compression dictionary into the table will attempt to acquire write locks. Please refer to the `ROWCOMPESTIMATE` option for details on dictionary insert locking. Commit processing will be done during the inspect processing. It is advisable to end the unit of work by issuing a `COMMIT` or `ROLLBACK` before starting the inspect operation.

Related reference:

- "db2Inspect - Inspect database for architectural integrity" on page 705

LIST APPLICATIONS

Displays to standard output the application program name, authorization ID (user name), application handle, application ID, and database name of all active database applications. This command can also optionally display an application's sequence number, status, status change time, and database path.

Scope:

This command only returns information for the database partition on which it is issued.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *sysmon*

Required connection:

Instance. To list applications for a remote instance, it is necessary to first attach to that instance.

Command syntax:

```
►►──LIST APPLICATIONS──┬──────────────────────────────────────┬──────────────────►
                       └─FOR──┬─DATABASE─┬──database-alias─┘
                              └─DB───────┘
```

```
►─┬───────────────────────────────────────┬──┬─────────────┬──────────────►◄
  ├─AT DBPARTITIONNUM──db-partition-number─┤  └─SHOW DETAIL─┘
  └─GLOBAL─────────────────────────────────┘
```

Command parameters:

FOR DATABASE database-alias
: Information for each application that is connected to the specified database is to be displayed. Database name information is not displayed. If this option is not specified, the command displays the information for each application that is currently connected to any database at the database partition to which the user is currently attached.

  The default application information is comprised of the following:

  - Authorization ID
  - Application program name
  - Application handle
  - Application ID
  - Database name.

AT DBPARTITIONNUM db-partition-number
: Specifies the database partition for which the status of the monitor switches is to be displayed.

GLOBAL
: Returns an aggregate result for all database partitions in a partitioned database system.

SHOW DETAIL
: Output will include the following additional information:

  - Sequence #
  - Application status
  - Status change time
  - Database path.

If this option is specified, it is recommended that the output be redirected to a file, and that the report be viewed with the help of an editor. The output lines might wrap around when displayed on the screen.

Examples:

To list detailed information about the applications connected to the SAMPLE database, issue:

```
list applications for database sample show detail
```

Usage notes:

The database administrator can use the output from this command as an aid to problem determination. In addition, this information is required if the database administrator wants to use the GET SNAPSHOT command or the FORCE APPLICATION command in an application.

To list applications at a remote instance (or a different local instance), it is necessary to first attach to that instance. If FOR DATABASE is specified when an attachment exists, and the database resides at an instance which differs from the current attachment, the command will fail.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODE can be substituted for DBPARTITIONNUM.

Related reference:
- "GET SNAPSHOT command" in *Command Reference*
- "FORCE APPLICATION command" in *Command Reference*
- "APPLICATIONS administrative view - Retrieve connected database application information" in *Administrative SQL Routines and Views*
- "FORCE APPLICATION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

LIST DATABASE PARTITION GROUPS

Lists all database partition groups associated with the current database.

Scope:

This command can be issued from any database partition that is listed in `$HOME/sqllib/db2nodes.cfg`. It returns the same information from any of these database partitions.

Authorization:

For the system catalogs `SYSCAT.DBPARTITIONGROUPS` and `SYSCAT.DBPARTITIONGROUPDEF`, one of the following is required:
- *sysadm* or *dbadm* authority
- CONTROL privilege
- SELECT privilege.

Required connection:

Database

Command syntax:

```
►►—LIST DATABASE PARTITION GROUPS—┬─────────────┬——————————————————————————►◄
                                   └─SHOW DETAIL─┘
```

Command parameters:

SHOW DETAIL
Specifies that the output should include the following information:
- Distribution map ID
- Database partition number
- In-use flag

Examples:

Following is sample output from the LIST DATABASE PARTITION GROUPS command:

```
DATABASE PARTITION GROUP NAME
-----------------------------
IBMCATGROUP
IBMDEFAULTGROUP

  2 record(s) selected.
```

Following is sample output from the LIST DATABASE PARTITION GROUPS SHOW DETAIL command:

```
DATABASE PARTITION GROUP NAME  PMAP_ID DATABASE PARTITION NUMBER IN_USE
------------------------------ ------- ------------------------- ------
IBMCATGROUP                          0                         0 Y
IBMDEFAULTGROUP                      1                         0 Y

  2 record(s) selected.
```

The fields are identified as follows:

DATABASE PARTITION GROUP NAME
The name of the database partition group. The name is repeated for each database partition in the database partition group.

PMAP_ID
The ID of the distribution map. The ID is repeated for each database partition in the database partition group.

DATABASE PARTITION NUMBER
The number of the database partition.

IN_USE
One of four values:

Y The database partition is being used by the database partition group.

D The database partition is going to be dropped from the database partition group as a result of a REDISTRIBUTE DATABASE PARTITION GROUP operation. When the operation completes, the database partition will not be included in reports from LIST DATABASE PARTITION GROUPS.

A The database partition has been added to the database partition group but is not yet added to the distribution map. The containers for the table spaces in the database partition group have been added on this database partition. The value is changed to `Y` when the REDISTRIBUTE DATABASE PARTITION GROUP operation completes successfully.

T The database partition has been added to the database partition group, but is not yet added to the distribution map. The containers for the table spaces in the database partition group have not been added on this database partition. Table space containers must be added on the new database partition for each table space in the database partition group. The value is changed to `A` when containers have successfully been added.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODEGROUPS can be substituted for DATABASE PARTITION GROUPS.

Related reference:
- "REDISTRIBUTE DATABASE PARTITION GROUP " on page 577

LIST PACKAGES/TABLES

Lists packages or tables associated with the current database.

Authorization:

For the system catalog `SYSCAT.PACKAGES` (LIST PACKAGES) and `SYSCAT.TABLES` (LIST TABLES), one of the following is required:
- *sysadm* or *dbadm* authority
- CONTROL privilege
- SELECT privilege.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax:

```
►►──LIST──┬─PACKAGES─┬──┬───────────────────────────────┬──┬─────────────┬──────────►◄
          └─TABLES───┘  │        ┌─USER───────────────┐ │  └─SHOW DETAIL─┘
                        └─FOR────┼─ALL────────────────┤─┘
                                 ├─SCHEMA──schema-name─┤
                                 └─SYSTEM─────────────┘
```

Command parameters:

FOR If the FOR clause is not specified, the packages or tables for USER are listed.

ALL Lists all packages or tables in the database.

SCHEMA
Lists all packages or tables in the database for the specified schema only.

SYSTEM
Lists all system packages or tables in the database.

USER Lists all user packages or tables in the database for the current user.

SHOW DETAIL
If this option is chosen with the LIST TABLES command, the full table name and schema name are displayed. If this option is not specified, the table name is truncated to 30 characters, and the ″>″ symbol in the 31st column represents the truncated portion of the table name; the schema name is truncated to 14 characters and the ″>″ symbol in the 15th column represents the truncated portion of the schema name. If this option is chosen with the LIST PACKAGES command, the full package schema (creator), version and boundby authid are displayed, and the package unique_id (consistency token shown in hexadecimal form). If this option is not specified, the schema name and bound by ID are truncated to 8 characters and the ″>″ symbol in the 9th column represents the truncated

portion of the schema or bound by ID; the version is truncated to 10 characters and the ">" symbol in the 11th column represents the truncated portion of the version.

Examples:

The following is sample output from LIST PACKAGES:

```
                              Bound     Total                         Isolation
Package    Schema    Version    by        sections     Valid  Format  level     Blocking
---------- --------- ---------- --------- ------------ ------ ------- --------- --------
F4INS      USERA     VER1       SNOWBELL           221 Y      0       CS        U
F4INS      USERA     VER2.0     SNOWBELL           201 Y      0       RS        U
F4INS      USERA     VER2.3     SNOWBELL           201 N      3       CS        U
F4INS      USERA     VER2.5     SNOWBELL           201 Y      0       CS        U
PKG12      USERA                USERA               12 Y      3       RR        B
PKG15      USERA                USERA               42 Y      3       RR        B
SALARY     USERT     YEAR2000   USERT               15 Y      3       CS        N
```

The following is sample output from LIST TABLES:

```
Table/View         Schema           Type       Creation time
------------------ ---------------- ---------- ----------------------------
DEPARTMENT         SMITH            T          1997-02-19-13.32.25.971890
EMP_ACT            SMITH            T          1997-02-19-13.32.27.851115
EMP_PHOTO          SMITH            T          1997-02-19-13.32.29.953624
EMP_RESUME         SMITH            T          1997-02-19-13.32.37.837433
EMPLOYEE           SMITH            T          1997-02-19-13.32.26.348245
ORG                SMITH            T          1997-02-19-13.32.24.478021
PROJECT            SMITH            T          1997-02-19-13.32.29.300304
SALES              SMITH            T          1997-02-19-13.32.42.973739
STAFF              SMITH            T          1997-02-19-13.32.25.156337

  9 record(s) selected.
```

Usage notes:

LIST PACKAGES and LIST TABLES commands are available to provide a quick interface to the system tables.

The following SELECT statements return information found in the system tables. They can be expanded to select the additional information that the system tables provide.

```
select tabname, tabschema, type, create_time
from syscat.tables
order by tabschema, tabname;

select pkgname, pkgschema, pkgversion, unique_id, boundby, total_sect,
   valid, format, isolation, blocking
from syscat.packages
order by pkgschema, pkgname, pkgversion;

select tabname, tabschema, type, create_time
from syscat.tables
where tabschema = 'SYSCAT'
order by tabschema, tabname;

select pkgname, pkgschema, pkgversion, unique_id, boundby, total_sect,
   valid, format, isolation, blocking
from syscat.packages
where pkgschema = 'NULLID'
order by pkgschema, pkgname, pkgversion;
```

```
select tabname, tabschema, type, create_time
from syscat.tables
where tabschema = USER
order by tabschema, tabname;

select pkgname, pkgschema, pkgversion, unique_id, boundby, total_sect,
   valid, format, isolation, blocking
from syscat.packages
where pkgschema = USER
order by pkgschema, pkgname, pkgversion;
```

Related concepts:
- "Catalog views" on page 161
- "Efficient SELECT statements" in *Performance Guide*

LIST TABLESPACE CONTAINERS

Lists containers for the specified table space.

The table space snapshot contains all of the information displayed by the LIST TABLESPACE CONTAINERS command.

Scope:

This command returns information only for the node on which it is executed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

Required connection:

Database

Command syntax:

```
►►──LIST TABLESPACE CONTAINERS FOR──tablespace-id──┬─────────────┬──────────────►◄
                                                   └─SHOW DETAIL─┘
```

Command parameters:

FOR tablespace-id
: An integer that uniquely represents a table space used by the current database. To get a list of all the table spaces used by the current database, use the LIST TABLESPACES command.

SHOW DETAIL
: If this option is not specified, only the following basic information about each container is provided:
  - Container ID
  - Name

- Type (file, disk, or path).

If this option is specified, the following additional information about each container is provided:

- Total number of pages
- Number of useable pages
- Accessible (yes or no).

Examples:

The following is sample output from LIST TABLESPACE CONTAINERS:

```
            Tablespace Containers for Tablespace 0

 Container ID                         = 0
 Name                                 = /home/smith/smith/NODE0000/
                                        SQL00001/SQLT0000.0
 Type                                 = Path
```

The following is sample output from LIST TABLESPACE CONTAINERS with SHOW DETAIL specified:

```
            Tablespace Containers for Tablespace 0

 Container ID                         = 0
 Name                                 = /home/smith/smith/NODE0000/
                                        SQL00001/SQLT0000.0
 Type                                 = Path
 Total pages                          = 895
 Useable pages                        = 895
 Accessible                           = Yes
```

Related concepts:

- "Snapshot monitor" in *System Monitor Guide and Reference*

Related reference:

- "sqlbtcq API - Get the query data for all table space containers" in *Administrative API Reference*
- "LIST TABLESPACES " on page 537
- "CONTAINER_UTILIZATION administrative view - Retrieve table space container and utilization information" in *Administrative SQL Routines and Views*
- "TBSP_UTILIZATION administrative view - Retrieve table space configuration and utilization information" in *Administrative SQL Routines and Views*

LIST TABLESPACES

Lists table spaces for the current database.

Information displayed by this command is also available in the table space snapshot.

Scope:

This command returns information only for the node on which it is executed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

Command syntax:

```
►►──LIST TABLESPACES──┬─────────────┬──────────────────────────────►◄
                      └─SHOW DETAIL─┘
```

Command parameters:

SHOW DETAIL

If this option is not specified, only the following basic information about each table space is provided:

- Table space ID
- Name
- Type (system managed space or database managed space)
- Contents (any data, long or index data, or temporary data)
- State, a hexadecimal value indicating the current table space state. The externally visible state of a table space is composed of the hexadecimal sum of certain state values. For example, if the state is ″quiesced: EXCLUSIVE″ and ″Load pending″, the value is `0x0004` + `0x0008`, which is `0x000c`. The db2tbst (Get Tablespace State) command can be used to obtain the table space state associated with a given hexadecimal value. Following are the bit definitions listed in `sqlutil.h`:

0x0	Normal
0x1	Quiesced: SHARE
0x2	Quiesced: UPDATE
0x4	Quiesced: EXCLUSIVE
0x8	Load pending
0x10	Delete pending
0x20	Backup pending
0x40	Roll forward in progress
0x80	Roll forward pending
0x100	Restore pending
0x100	Recovery pending (not used)
0x200	Disable pending
0x400	Reorg in progress
0x800	Backup in progress
0x1000	Storage must be defined
0x2000	Restore in progress
0x4000	Offline and not accessible
0x8000	Drop pending
0x2000000	Storage may be defined
0x4000000	StorDef is in 'final' state
0x8000000	StorDef was changed prior to rollforward
0x10000000	DMS rebalancer is active
0x20000000	TBS deletion in progress
0x40000000	TBS creation in progress
0x8	For service use only

If this option is specified, the following additional information about each table space is provided:

- Total number of pages
- Number of usable pages
- Number of used pages
- Number of free pages
- High water mark (in pages)
- Page size (in bytes)
- Extent size (in pages)
- Prefetch size (in pages)
- Number of containers
- Minimum recovery time (displayed only if not zero)
- State change table space ID (displayed only if the table space state is ″load pending″ or ″delete pending″)
- State change object ID (displayed only if the table space state is ″load pending″ or ″delete pending″)
- Number of quiescers (displayed only if the table space state is ″quiesced: SHARE″, ″quiesced: UPDATE″, or ″quiesced: EXCLUSIVE″)
- Table space ID and object ID for each quiescer (displayed only if the number of quiescers is greater than zero).

Examples:

The following are two sample outputs from LIST TABLESPACES SHOW DETAIL.

```
          Tablespaces for Current Database
 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = Database managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 895
 Useable pages                        = 895
 Used pages                           = 895
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
 Tablespace ID                        = 2
 Name                                 = USERSPACE1
 Type                                 = Database managed space
 Contents                             = Any data
 State                                = 0x000c
   Detailed explanation:
     Quiesced: EXCLUSIVE
     Load pending
 Total pages                          = 337
 Useable pages                        = 337
 Used pages                           = 337
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
 State change tablespace ID           = 2
 State change object ID               = 3
 Number of quiescers                  = 1
   Quiescer 1:
     Tablespace ID                    = 2
     Object ID                        = 3
DB21011I  In a partitioned database server environment, only the table spaces
on the current node are listed.

           Tablespaces for Current Database
 Tablespace ID                        = 0
 Name                                 = SYSCATSPACE
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1200
 Useable pages                        = 1200
 Used pages                           = 1200
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
 Tablespace ID                        = 1
 Name                                 = TEMPSPACE1
 Type                                 = System managed space
 Contents                             = Temporary data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)              = Not applicable
 Page size (bytes)                    = 4096
 Extent size (pages)                  = 32
 Prefetch size (pages)                = 32
 Number of containers                 = 1
Tablespace ID                         = 2
 Name                                 = USERSPACE1
 Type                                 = System managed space
 Contents                             = Any data
 State                                = 0x0000
   Detailed explanation:
     Normal
 Total pages                          = 1
 Useable pages                        = 1
 Used pages                           = 1
 Free pages                           = Not applicable
 High water mark (pages)             = Not applicable
 Page size (bytes)                   = 4096
 Extent size (pages)                 = 32
 Prefetch size (pages)               = 32
 Number of containers                = 1
Tablespace ID                        = 3
 Name                                = DMS8K
 Type                                = Database managed space
 Contents                            = Any data
 State                               = 0x0000
   Detailed explanation:
     Normal
 Total pages                         = 2000
 Useable pages                       = 1952
 Used pages                          = 96
 Free pages                          = 1856
 High water mark (pages)             = 96
 Page size (bytes)                   = 8192
 Extent size (pages)                 = 32
 Prefetch size (pages)               = 32
 Number of containers                = 2
Tablespace ID                        = 4
 Name                                = TEMP8K
 Type                                = System managed space
 Contents                            = Temporary data
 State                               = 0x0000
   Detailed explanation:
     Normal
 Total pages                         = 1
 Useable pages                       = 1
 Used pages                          = 1
 Free pages                          = Not applicable
 High water mark (pages)             = Not applicable
 Page size (bytes)                   = 8192
 Extent size (pages)                 = 32
 Prefetch size (pages)               = 32
 Number of containers                = 1
DB21011I  In a partitioned database server environment, only the table spaces
on the current node are listed.
```

Usage notes:

In a partitioned database environment, this command does not return all the table spaces in the database. To obtain a list of all the table spaces, query SYSCAT.TABLESPACES.

During a table space rebalance, the number of usable pages includes pages for the newly added container, but these new pages are not reflected in the number of free pages until the rebalance is complete. When a table space rebalance is not in progress, the number of used pages plus the number of free pages equals the number of usable pages.

Related reference:

- "sqlbmtsq - Get the query data for all table spaces" on page 783
- "LIST TABLESPACE CONTAINERS " on page 536
- "db2tbst - Get table space state command" in *Command Reference*
- "CONTAINER_UTILIZATION administrative view - Retrieve table space container and utilization information" in *Administrative SQL Routines and Views*
- "TBSP_UTILIZATION administrative view - Retrieve table space configuration and utilization information" in *Administrative SQL Routines and Views*

LOAD

Loads data into a DB2 table. Data residing on the server can be in the form of a file, tape, or named pipe. If the `COMPRESS` attribute for the table is set to `YES`, the data loaded will be subject to compression on every data and database partition for which a dictionary already exists in the table.

Restrictions:

The load utility does not support loading data at the hierarchy level. The load utility is not compatible with range-clustered tables.

Scope:

This command can be issued against multiple database partitions in a single request.

Authorization:

One of the following:

- *sysadm*
- *dbadm*
- load authority on the database and
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.
- To load data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the load fails and an error (SQLSTATE 5U014) is returned.
- To load data into a table that has protected rows, the session authorization id must hold a security label that meets these criteria:
  - It is part of the security policy protecting the table
  - It was granted to the session authorization ID for write access or for all access

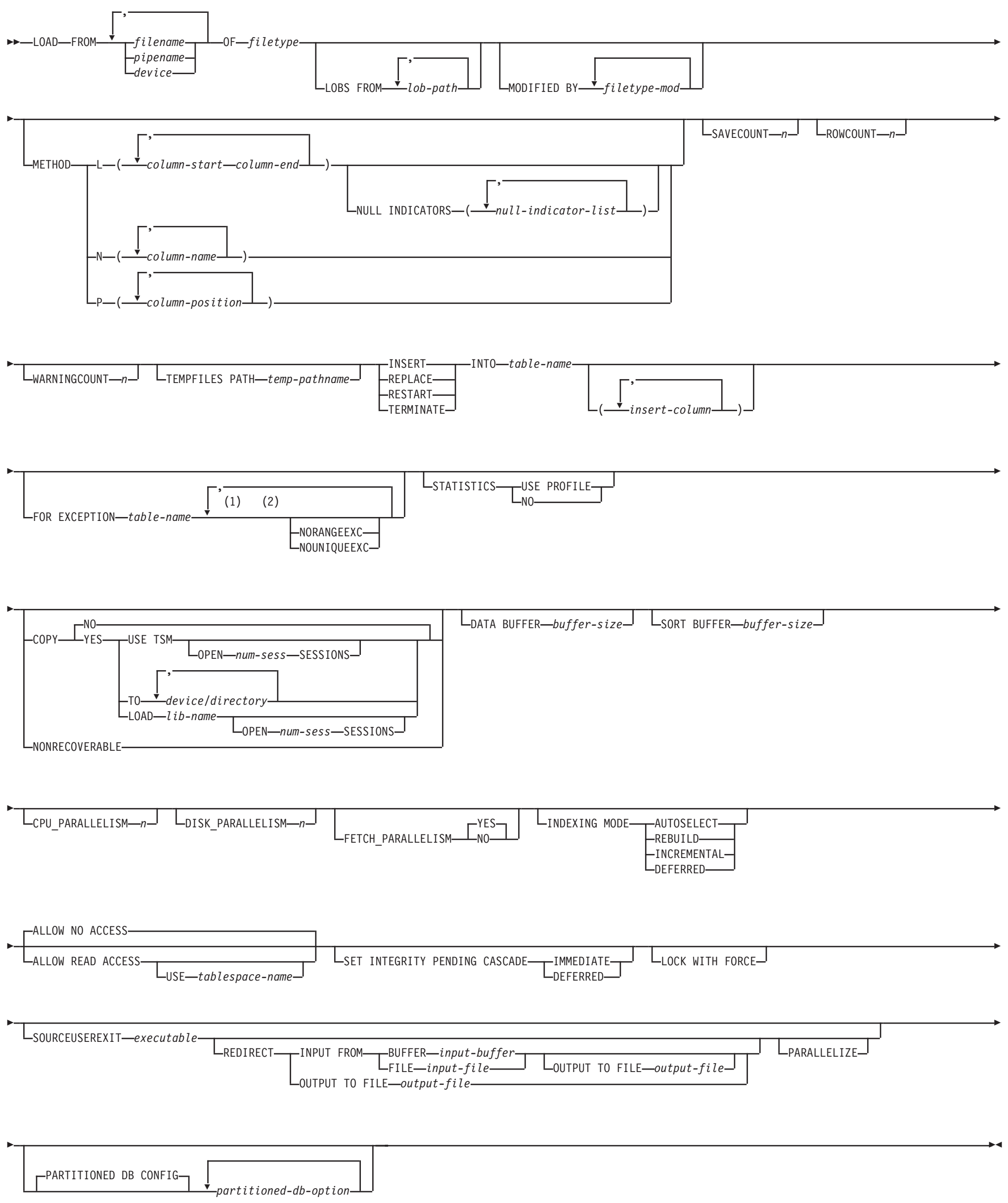
  If the session authorization id does not hold such a security label then the load fails and an error (SQLSTATE 5U014) is returned. This security label is used to protect a loaded row if the session authorization ID's LBAC credentials do not allow it to write to the security label that protects that row in the data. This does not happen, however, when the security policy protecting the table was created with the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option of the CREATE SECURITY POLICY statement. In this case the load fails and an error (SQLSTATE 42519) is returned.
- If the REPLACE option is specified, the session authorization ID must have the authority to drop the table.

Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by the instance owner, regardless of who invokes the command.

Required connection:

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

Command syntax:


LOAD FROM filename pipename device OF filetype
LOBS FROM lob-path
MODIFIED BY filetype-mod
METHOD L (column-start column-end)
NULL INDICATORS (null-indicator-list)
N (column-name)
P (column-position)
SAVECOUNT n
ROWCOUNT n
WARNINGCOUNT n
TEMPFILES PATH temp-pathname
INSERT REPLACE RESTART TERMINATE INTO table-name
(insert-column)
FOR EXCEPTION table-name (1) (2) NORANGEEXC NOUNIQUEEXC
STATISTICS USE PROFILE NO
COPY NO YES USE TSM OPEN num-sess SESSIONS
TO device/directory
LOAD lib-name OPEN num-sess SESSIONS
NONRECOVERABLE
DATA BUFFER buffer-size
SORT BUFFER buffer-size
CPU_PARALLELISM n
DISK_PARALLELISM n
FETCH_PARALLELISM YES NO
INDEXING MODE AUTOSELECT REBUILD INCREMENTAL DEFERRED
ALLOW NO ACCESS
ALLOW READ ACCESS USE tablespace-name
SET INTEGRITY PENDING CASCADE IMMEDIATE DEFERRED
LOCK WITH FORCE
SOURCEUSEREXIT executable
REDIRECT INPUT FROM BUFFER input-buffer FILE input-file
OUTPUT TO FILE output-file
OUTPUT TO FILE output-file
PARALLELIZE
PARTITIONED DB CONFIG partitioned-db-option

Notes:

1 These keywords can appear in any order.

2 Each of these keywords can only appear once.

Command parameters:

FROM filename/pipename/device/

Notes:

1. If data is exported into a file using the EXPORT command using the ADMIN_CMD procedure, the data file is owned by the fenced user ID. This file is not usually accessible by the instance owner. To run the LOAD from CLP or the ADMIN_CMD procedure, the data file must be accessible by the instance owner ID, so read access to the data file must be granted to the instance owner.
2. Loading data from multiple IXF files is supported if the files are physically separate, but logically one file. It is *not* supported if the files are both logically and physically separate. (Multiple physical files would be considered logically one if they were all created with one invocation of the EXPORT command.)

OF filetype

Specifies the format of the data:

- ASC (non-delimited ASCII format)
- DEL (delimited ASCII format)
- IXF (integrated exchange format, PC version), exported from the same or from another DB2 table
- CURSOR (a cursor declared against a SELECT or VALUES statement).

LOBS FROM lob-path

The path to the data files containing LOB values to be loaded. The path must end with a slash (/). The names of the LOB data files are stored in the main data file (ASC, DEL, or IXF), in the column that will be loaded into the LOB column. The maximum number of paths that can be specified is 999. This will implicitly activate the `LOBSINFILE` behaviour.

This option is ignored when specified in conjunction with the CURSOR filetype.

MODIFIED BY filetype-mod

Specifies file type modifier options. See File type modifiers for the load utility.

METHOD

L
Specifies the start and end column numbers from which to load data. A column number is a byte offset from the beginning of a row of data. It is numbered starting from 1. This method can only be used with ASC files, and is the only valid method for that file type.

NULL INDICATORS null-indicator-list

This option can only be used when the METHOD L parameter is specified; that is, the input file is an ASC file). The null indicator list is a comma-separated list of positive integers specifying the column number of each null indicator field. The column number is the byte offset of the null indicator field from the beginning of a row of data. There must be one entry in the null indicator list for each data field defined in the METHOD L parameter. A column number of zero indicates that the corresponding data field always contains data.

A value of `Y` in the NULL indicator column specifies that the column data is NULL. Any character *other than* `Y` in the NULL indicator column specifies that the column data is not NULL, and that column data specified by the METHOD L option will be loaded.

The NULL indicator character can be changed using the MODIFIED BY option.

N Specifies the names of the columns in the data file to be loaded. The case of these column names must match the case of the corresponding names in the system catalogs. Each table column that is not nullable should have a corresponding entry in the METHOD N list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method N (F2, F1, F4, F3)` is a valid request, while `method N (F2, F1)` is not valid. This method can only be used with file types IXF or CURSOR.

P Specifies the field numbers (numbered from 1) of the input data fields to be loaded. Each table column that is not nullable should have a corresponding entry in the METHOD P list. For example, given data fields F1, F2, F3, F4, F5, and F6, and table columns C1 INT, C2 INT NOT NULL, C3 INT NOT NULL, and C4 INT, `method P (2, 1, 4, 3)` is a valid request, while `method P (2, 1)` is not valid. This method can only be used with file types IXF, DEL, or CURSOR, and is the only valid method for the DEL file type.

SAVECOUNT n
Specifies that the load utility is to establish consistency points after every *n* rows. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using LOAD QUERY. If the value of *n* is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

The default value is zero, meaning that no consistency points will be established, unless necessary.

This option is ignored when specified in conjunction with the CURSOR filetype.

ROWCOUNT n
Specifies the number of *n* physical records in the file to be loaded. Allows a user to load only the first *n* rows in a file.

WARNINGCOUNT n
Stops the load operation after *n* warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If *n* is zero, or this option is not specified, the load operation will continue regardless of the number of warnings issued. If the load operation is stopped because the threshold of warnings was encountered, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

TEMPFILES PATH temp-pathname
Specifies the name of the path to be used when creating temporary files during a load operation, and should be fully qualified according to the server database partition.

Temporary files take up file system space. Sometimes, this space requirement is quite substantial. Following is an estimate of how much file system space should be allocated for all temporary files:

- 136 bytes for each message that the load utility generates
- 15KB overhead if the data file contains long field data or LOBs. This quantity can grow significantly if the INSERT option is specified, and there is a large amount of long field or LOB data already in the table.

INSERT
One of four modes under which the load utility can execute. Adds the loaded data to the table without changing the existing table data.

REPLACE
One of four modes under which the load utility can execute. Deletes all existing data from the table, and inserts the loaded data. The table definition and index definitions are not changed. If this option is used when moving data between hierarchies, only the data for an entire hierarchy, not individual subtables, can be replaced.

RESTART
One of four modes under which the load utility can execute. Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

TERMINATE
One of four modes under which the load utility can execute. Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects might be marked as invalid, in which case index rebuild will automatically take place at next access). If the load operation being terminated is a load REPLACE, the table will be truncated to an empty table after the load TERMINATE operation. If the load operation being terminated is a load INSERT, the table will retain all of its original records after the load TERMINATE operation.

The load terminate option will not remove a backup pending state from table spaces.

INTO table-name
Specifies the database table into which the data is to be loaded. This table cannot be a system table or a declared temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form *schema.tablename*. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

insert-column
Specifies the table column into which the data is to be inserted.

The load utility cannot parse columns whose names contain one or more spaces. For example, will fail because of the `Int 4` column. The solution is to enclose such column names with double quotation marks:

FOR EXCEPTION table-name
Specifies the exception table into which rows in error will be copied. Any row that is in violation of a unique index or a primary key index is copied. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

Information that is written to the exception table is *not* written to the dump file. In a partitioned database environment, an exception table must be defined for those database partitions on which the loading table is defined. The dump file, on the other hand, contains rows that cannot be loaded because they are invalid or have syntax errors.

NORANGEEXC
Indicates that if a row is rejected because of a range violation it will not be inserted into the exception table.

NOUNIQUEEXC
Indicates that if a row is rejected because it violates a unique constraint it will not be inserted into the exception table.

STATISTICS USE PROFILE
Instructs load to collect statistics during the load according to the profile defined for this table. This profile must be created before load is executed. The profile is created by the RUNSTATS command. If the profile does not exist and load is instructed to collect statistics according to the profile, a warning is returned and no statistics are collected.

STATISTICS NO
Specifies that no statistics are to be collected, and that the statistics in the catalogs are not to be altered. This is the default.

COPY NO
Specifies that the table space in which the table resides will be placed in backup pending state if forward recovery is enabled (that is, *logretain* or *userexit* is on). The COPY NO option will also put the table space state into the Load in Progress table space state. This is a transient state that will disappear when the load completes or aborts. The data in any table in the table space cannot be updated or deleted until a table space backup or a full database backup is made. However, it is possible to access the data in any table by using the SELECT statement.

LOAD with COPY NO on a recoverable database leaves the table spaces in a backup pending state. For example, performing a LOAD with COPY NO and INDEXING MODE DEFERRED will leave indexes needing a refresh. Certain queries on the table might require an index scan and will not succeed until the indexes are refreshed. The index cannot be refreshed if it resides in a table space which is in the backup pending state. In that case, access to the table will not be allowed until a backup is taken. Index refresh is done automatically by the database when the index is accessed by a query.

COPY YES
Specifies that a copy of the loaded data will be saved. This option is invalid if forward recovery is disabled (both *logretain* and *userexit* are off).

USE TSM
Specifies that the copy will be stored using Tivoli Storage Manager (TSM).

OPEN num-sess SESSIONS
The number of I/O sessions to be used with TSM or the vendor product. The default value is 1.

TO device/directory
Specifies the device or directory on which the copy image will be created.

LOAD lib-name
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. It can contain the full path. If the full path is not given, it will default to the path where the user exit programs reside.

NONRECOVERABLE
Specifies that the load transaction is to be marked as non-recoverable and that it will not be possible to recover it by a subsequent roll forward action. The roll forward utility will skip the transaction and will mark the table into which data was being loaded as "invalid". The utility will also ignore any subsequent transactions against that table. After the roll forward operation is completed, such a table can only be dropped or restored from a backup (full or table space) taken after a commit point following the completion of the non-recoverable load operation.

With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation.

WITHOUT PROMPTING
Specifies that the list of data files contains all the files that are to be loaded, and that the devices or directories listed are sufficient for the entire load operation. If a continuation input file is not found, or the copy targets are filled before the load operation finishes, the load operation will fail, and the table will remain in load pending state.

DATA BUFFER buffer-size
Specifies the number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the minimum required resource is used, and no warning is returned.

This memory is allocated directly from the utility heap, whose size can be modified through the *util_heap_sz* database configuration parameter.

If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

SORT BUFFER buffer-size
This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the INDEXING MODE parameter is not specified as DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory that is used when loading tables with many indexes without changing the value of SORTHEAP, which would also affect general query processing.

CPU_PARALLELISM n
Specifies the number of processes or threads that the load utility will

spawn for parsing, converting, and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, or has not been specified, the load utility uses an intelligent default value (usually based on the number of CPUs available) at run time.

Notes:

1. If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs or the value specified by the user.
2. Specifying a small value for the SAVECOUNT parameter causes the loader to perform many more I/O operations to flush both data and table metadata. When CPU_PARALLELISM is greater than one, the flushing operations are asynchronous, permitting the loader to exploit the CPU. When CPU_PARALLELISM is set to one, the loader waits on I/O during consistency points. A load operation with CPU_PARALLELISM set to two, and SAVECOUNT set to 10 000, completes faster than the same operation with CPU_PARALLELISM set to one, even though there is only one CPU.

DISK_PARALLELISM n
Specifies the number of processes or threads that the load utility will spawn for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

FETCH_PARALLELISM YES/NO
When performing a load from a cursor where the cursor is declared using the `DATABASE` keyword, or when using the API `sqlu_remotefetch_entry` media entry, and this option is set to `YES`, the load utility attempts to parallelize fetching from the remote data source if possible. If set to `NO`, no parallel fetching is performed. The default value is `YES`. For more information, see Moving data using the CURSOR file type.

INDEXING MODE
Specifies whether the load utility is to rebuild indexes or to extend them incrementally. Valid values are:

AUTOSELECT
The load utility will automatically decide between REBUILD or INCREMENTAL mode. The decision is based on the amount of data being loaded and the depth of the index tree. Information relating to the depth of the index tree is stored in the index object. RUNSTATS is not required to populate this information. AUTOSELECT is the default indexing mode.

REBUILD
All indexes will be rebuilt. The utility must have sufficient resources to sort all index key parts for both old and appended table data.

INCREMENTAL
Indexes will be extended with new data. This approach consumes index free space. It only requires enough sort space to append index keys for the inserted records. This method is only supported in cases where the index object is valid and accessible at the start of a load operation (it is, for example, not valid immediately following a load operation in which the DEFERRED mode was

specified). If this mode is specified, but not supported due to the state of the index, a warning is returned, and the load operation continues in REBUILD mode. Similarly, if a load restart operation is begun in the load build phase, INCREMENTAL mode is not supported.

Incremental indexing is not supported when all of the following conditions are true:

- The LOAD COPY option is specified (*logarchmeth1* with the `USEREXIT` or `LOGRETAIN` option).
- The table resides in a DMS table space.
- The index object resides in a table space that is shared by other table objects belonging to the table being loaded.

To bypass this restriction, it is recommended that indexes be placed in a separate table space.

DEFERRED

The load utility will not attempt index creation if this mode is specified. Indexes will be marked as needing a refresh. The first access to such indexes that is unrelated to a load operation might force a rebuild, or indexes might be rebuilt when the database is restarted. This approach requires enough sort space for all key parts for the largest index. The total time subsequently taken for index construction is longer than that required in REBUILD mode. Therefore, when performing multiple load operations with deferred indexing, it is advisable (from a performance viewpoint) to let the last load operation in the sequence perform an index rebuild, rather than allow indexes to be rebuilt at first non-load access.

Deferred indexing is only supported for tables with non-unique indexes, so that duplicate keys inserted during the load phase are not persistent after the load operation.

ALLOW NO ACCESS

Load will lock the target table for exclusive access during the load. The table state will be set to Load In Progress during the load. ALLOW NO ACCESS is the default behavior. It is the only valid option for LOAD REPLACE.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending state.

ALLOW READ ACCESS

Load will lock the target table in a share mode. The table state will be set to both Load In Progress and Read Access. Readers can access the non-delta portion of the data while the table is being load. In other words, data that existed before the start of the load will be accessible by readers to the table, data that is being loaded is not available until the load is complete. LOAD TERMINATE or LOAD RESTART of an ALLOW READ ACCESS load can use this option; LOAD TERMINATE or LOAD RESTART of an ALLOW NO ACCESS load cannot use this option. Furthermore, this option is not valid if the indexes on the target table are marked as requiring a rebuild.

When there are constraints on the table, the table state will be set to Set Integrity Pending as well as Load In Progress, and Read Access. At the end of the load, the table state Load In Progress will be removed but the table states Set Integrity Pending and Read Access will remain. The SET INTEGRITY statement must be used to take the table out of Set Integrity Pending. While the table is in Set Integrity Pending and Read Access states, the non-delta portion of the data is still accessible to readers, the new (delta) portion of the data will remain inaccessible until the SET INTEGRITY statement has completed. A user can perform multiple loads on the same table without issuing a SET INTEGRITY statement. Only the original (checked) data will remain visible, however, until the SET INTEGRITY statement is issued.

ALLOW READ ACCESS also supports the following modifiers:

USE tablespace-name

If the indexes are being rebuilt, a shadow copy of the index is built in table space *tablespace-name* and copied over to the original table space at the end of the load during an INDEX COPY PHASE. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same table space as the index object. If the shadow copy is created in the same table space as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different table space from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load during the INDEX COPY PHASE.

Without this option the shadow index is built in the same table space as the original. Since both the original index and shadow index by default reside in the same table space simultaneously, there might be insufficient space to hold both indexes within one table space. Using this option ensures that you retain enough table space for the indexes.

This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

SET INTEGRITY PENDING CASCADE

If LOAD puts the table into Set Integrity Pending state, the SET INTEGRITY PENDING CASCADE option allows the user to specify whether or not Set Integrity Pending state of the loaded table is immediately cascaded to all descendents (including descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables).

IMMEDIATE

Indicates that Set Integrity Pending state is immediately extended to all descendent foreign key tables, descendent immediate materialized query tables and descendent staging tables. For a LOAD INSERT operation, Set Integrity Pending state is not extended to descendent foreign key tables even if the IMMEDIATE option is specified.

When the loaded table is later checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement), descendent foreign key tables that were placed in Set Integrity Pending Read Access state will be put into Set Integrity Pending No Access state.

DEFERRED

Indicates that only the loaded table will be placed in the Set Integrity Pending state. The states of the descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged.

Descendent foreign key tables might later be implicitly placed in Set Integrity Pending state when their parent tables are checked for constraint violations (using the IMMEDIATE CHECKED option of the SET INTEGRITY statement). Descendent immediate materialized query tables and descendent immediate staging tables will be implicitly placed in Set Integrity Pending state when one of its underlying tables is checked for integrity violations. A warning (SQLSTATE 01586) will be issued to indicate that dependent tables have been placed in Set Integrity Pending state. See the Notes section of the SET INTEGRITY statement in the SQL Reference for when these descendent tables will be put into Set Integrity Pending state.

If the SET INTEGRITY PENDING CASCADE option is not specified:

- Only the loaded table will be placed in Set Integrity Pending state. The state of descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables will remain unchanged, and can later be implicitly put into Set Integrity Pending state when the loaded table is checked for constraint violations.

If LOAD does not put the target table into Set Integrity Pending state, the SET INTEGRITY PENDING CASCADE option is ignored.

LOCK WITH FORCE

The utility acquires various locks including table locks in the process of loading. Rather than wait, and possibly timeout, when acquiring a lock, this option allows load to force off other applications that hold conflicting locks on the target table. Applications holding conflicting locks on the system catalog tables will not be forced off by the load utility. Forced applications will roll back and release the locks the load utility needs. The load utility can then proceed. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

ALLOW NO ACCESS loads might force applications holding conflicting locks at the start of the load operation. At the start of the load the utility can force applications that are attempting to either query or modify the table.

ALLOW READ ACCESS loads can force applications holding conflicting locks at the start or end of the load operation. At the start of the load the load utility can force applications that are attempting to modify the table. At the end of the load operation, the load utility can force applications that are attempting to either query or modify the table.

SOURCEUSEREXIT*executable*

Specifies an executable filename which will be called to feed data into the utility.

REDIRECT

INPUT FROM

BUFFER *input-buffer*

The stream of bytes specified in *input-buffer* is passed into the STDIN file descriptor of the process executing the given executable.

FILE *input-file*
The contents of this client-side file are passed into the STDIN file descriptor of the process executing the given executable.

OUTPUT TO

FILE *output-file*
The STDOUT and STDERR file descriptors are captured to the fully qualified server-side file specified.

PARALLELIZE
Increases the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable in multi-partition database environments and is ingored in single-partition database enviroments.

For more information, see Moving data using a customized application (user exit).

PARTITIONED DB CONFIG
Allows you to execute a load into a table distributed across multiple database partitions. The PARTITIONED DB CONFIG parameter allows you to specify partitioned database-specific configuration options. The `partitioned-db-option` values can be any of the following:

```
PART_FILE_LOCATION x
OUTPUT_DBPARTNUMS x
PARTITIONING_DBPARTNUMS x
MODE x
MAX_NUM_PART_AGENTS x
ISOLATE_PART_ERRS x
STATUS_INTERVAL x
PORT_RANGE x
CHECK_TRUNCATION
MAP_FILE_INPUT x
MAP_FILE_OUTPUT x
TRACE x
NEWLINE
DISTFILE x
OMIT_HEADER
RUN_STAT_DBPARTNUM x
```

Detailed descriptions of these options are provided in Load configuration options for partitioned database environments.

RESTARTCOUNT
Reserved.

USING directory
Reserved.

Usage notes:

- Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted.
- The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update materialized query

tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in Set Integrity Pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in Set Integrity Pending state. Issue the SET INTEGRITY statement to take the tables out of Set Integrity Pending state. Load operations cannot be carried out on replicated materialized query tables.

- If a clustering index exists on the table, the data should be sorted on the clustering index prior to loading. Data does not need to be sorted prior to loading into a multidimensional clustering (MDC) table, however.
- If you specify an exception table when loading into a protected table, any rows that are protected by invalid security labels will be sent to that table. This might allow users that have access to the exception table to access to data that they would not normally be authorized to access. For better security be careful who you grant exception table access to, delete each row as soon as it is repaired and copied to the table being loaded, and drop the exception table as soon as you are done with it.
- Security labels in their internal format might contain newline characters. If you load the file using the DEL file format, those newline characters can be mistaken for delimiters. If you have this problem use the older default priority for delimiters by specifying the delprioritychar file type modifier in the LOAD command.
- For performing a load using the `CURSOR` filetype where the `DATABASE` keyword was specified during the `DECLARE CURSOR` command, the user ID and password used to authenticate against the database currently connected to (for the load) will be used to authenticate against the source database (specified by the `DATABASE` option of the `DECLARE CURSOR` command). If no user ID or password was specified for the connection to the loading database, a user ID and password for the source database must be specified during the `DECLARE CURSOR` command.

Related concepts:

- “Load overview” in *Data Movement Utilities Guide and Reference*
- “Privileges, authorities, and authorizations required to use Load” on page 414

Related tasks:

- “Loading data” in *Data Movement Utilities Guide and Reference*

Related reference:

- “QUIESCE TABLESPACES FOR TABLE ” on page 569
- “LOAD command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “Load - CLP examples” in *Data Movement Utilities Guide and Reference*
- “Load configuration options for partitioned database environments” in *Data Movement Utilities Guide and Reference*

File type modifiers for the load utility

Table 123. Valid file type modifiers for the load utility: All file formats

Modifier	**Description**
anyorder	This modifier is used in conjunction with the *cpu_parallelism* parameter. Specifies that the preservation of source data order is not required, yielding significant additional performance benefit on SMP systems. If the value of *cpu_parallelism* is 1, this option is ignored. This option is not supported if SAVECOUNT > 0, since crash recovery after a consistency point requires that data be loaded in sequence.
generatedignore	This modifier informs the load utility that data for all generated columns is present in the data file but should be ignored. This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedmissing` or the `generatedoverride` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated column (not even NULLs). This results in all generated column values being generated by the utility. This modifier cannot be used with either the `generatedignore` or the `generatedoverride` modifier.

generatedoverride	This modifier instructs the load utility to accept user-supplied data for all generated columns in the table (contrary to the normal rules for these types of columns). This is useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for a non-nullable generated column will be rejected (SQL3116W). When this modifier is used, the table will be placed in Set Integrity Pending state. To take the table out of Set Integrity Pending state without verifying the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > GENERATED COLUMN`
	`  IMMEDIATE UNCHECKED`
	
	To take the table out of Set Integrity Pending state and force verification of the user-supplied values, issue the following command after the load operation:
	`SET INTEGRITY FOR < table-name > IMMEDIATE CHECKED.`
	
	When this modifier is specified and there is a generated column in any of the partitioning keys, dimension keys or distribution keys, then the `LOAD` command will automatically convert the modifier to `generatedignore` and proceed with the load. This will have the effect of regenerating all of the generated column values.
	
	This modifier cannot be used with either the `generatedmissing` or the `generatedignore` modifier.

identityignore	This modifier informs the load utility that data for the identity column is present in the data file but should be ignored. This results in all identity values being generated by the utility. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This means that for GENERATED ALWAYS columns, no rows will be rejected. This modifier cannot be used with either the `identitymissing` or the `identityoverride` modifier.
identitymissing	If this modifier is specified, the utility assumes that the input data file contains no data for the identity column (not even NULLs), and will therefore generate a value for each row. The behavior will be the same for both GENERATED ALWAYS and GENERATED BY DEFAULT identity columns. This modifier cannot be used with either the `identityignore` or the `identityoverride` modifier.

Table 123. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
identityoverride	This modifier should be used only when an identity column defined as GENERATED ALWAYS is present in the table to be loaded. It instructs the utility to accept explicit, non-NULL data for such a column (contrary to the normal rules for these types of identity columns). This is useful when migrating data from another database system when the table must be defined as GENERATED ALWAYS, or when loading a table from data that was recovered using the DROPPED TABLE RECOVERY option on the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data or NULL data for the identity column will be rejected (SQL3116W). This modifier cannot be used with either the `identitymissing` or the `identityignore` modifier. The load utility will not attempt to maintain or verify the uniqueness of values in the table's identity column when this option is used.
indexfreespace=*x*	*x* is an integer between 0 and 99 inclusive. The value is interpreted as the percentage of each index page that is to be left as free space when load rebuilds the index. Load with INDEXING MODE INCREMENTAL ignores this option. The first entry in a page is added without restriction; subsequent entries are added the percent free space threshold can be maintained. The default value is the one used at CREATE INDEX time. This value takes precedence over the PCTFREE value specified in the CREATE INDEX statement; the registry variable DB2 INDEX FREE takes precedence over indexfreespace. The indexfreespace option affects index leaf pages only.
lobsinfile	*lob-path* specifies the path to the files containing LOB data. The ASC, DEL, or IXF load input files contain the names of the files having LOB data in the LOB column. This option is not supported in conjunction with the CURSOR filetype. The LOBS FROM clause specifies where the LOB files are located when the "lobsinfile" modifier is used. The LOBS FROM clause will implicitly activate the LOBSINFILE behavior. The LOBS FROM clause conveys to the LOAD utility the list of paths to search for the LOB files while loading the data. Each path contains at least one file that contains at least one LOB pointed to by a Lob Location Specifier (LLS) in the data file. The LLS is a string representation of the location of a LOB in a file stored in the LOB file path. The format of an LLS is *filename.ext.nnn.mmm/*, where *filename.ext* is the name of the file that contains the LOB, *nnn* is the offset in bytes of the LOB within the file, and *mmm* is the length of the LOB in bytes. For example, if the string `db2exp.001.123.456/` is stored in the data file, the LOB is located at offset 123 in the file db2exp.001, and is 456 bytes long. To indicate a null LOB , enter the size as -1. If the size is specified as 0, it is treated as a 0 length LOB. For null LOBS with length of -1, the offset and the file name are ignored. For example, the LLS of a null LOB might be db2exp.001.7.-1/.
noheader	Skips the header verification code (applicable only to load operations into tables that reside in a single-partition database partition group). If the default MPP load (mode PARTITION_AND_LOAD) is used against a table residing in a single-partition database partition group, the file is not expected to have a header. Thus the noheader modifier is not needed. If the LOAD_ONLY mode is used, the file is expected to have a header. The only circumstance in which you should need to use the noheader modifier is if you wanted to perform LOAD_ONLY operation using a file that does not have a header.
norowwarnings	Suppresses all warnings about rejected rows.

Table 123. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
pagefreespace=*x*	*x* is an integer between 0 and 100 inclusive. The value is interpreted as the percentage of each data page that is to be left as free space. If the specified value is invalid because of the minimum row size, (for example, a row that is at least 3 000 bytes long, and an *x* value of 50), the row will be placed on a new page. If a value of 100 is specified, each row will reside on a new page. The PCTFREE value of a table determines the amount of free space designated per page. If a pagefreespace value on the load operation or a PCTFREE value on a table have not been set, the utility will fill up as much space as possible on each page. The value set by pagefreespace overrides the PCTFREE value specified for the table.

seclabelchar	Indicates that security labels in the input source file are in the string format for security label values rather than in the default encoded numeric format. LOAD converts each security label into the internal format as it is loaded. If a string is not in the proper format the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3242W) is returned. If the string does not represent a valid security label that is part of the security policy protecting the table then the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3243W) is returned.
	
	This modifier cannot be specified if the seclabelname modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might look like this:
	`"CONFIDENTIAL:ALPHA:G2"`
	`"CONFIDENTIAL;SIGMA:G2"`
	`"TOP SECRET:ALPHA:G2"`
	
	To load or import this data, the SECLABELCHAR file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELCHAR INSERT INTO t1`

seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. LOAD will convert the name to the appropriate security label if it exists. If no security label exists with the indicated name for the security policy protecting the table the row is not loaded and a warning (SQLSTATE 01H53, SQLCODE SQL3244W) is returned.
	
	This modifier cannot be specified if the seclabelchar modifier is specified, otherwise the load fails and an error (SQLCODE SQL3525N) is returned.
	
	If you have a table consisting of a single `DB2SECURITYLABEL` column, the data file might consist of security label names similar to:
	`"LABEL1"`
	`"LABEL1"`
	`"LABEL2"`
	
	To load or import this data, the `SECLABELNAME` file type modifier must be used:
	`LOAD FROM input.del OF DEL MODIFIED BY SECLABELNAME INSERT INTO t1`
	
	Note: If the file type is ASC, any spaces following the name of the security label will be interpreted as being part of the name. To avoid this use the striptblanks file type modifier to make sure the spaces are removed.

Table 123. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
totalfreespace=*x*	*x* is an integer greater than or equal to 0 . The value is interpreted as the percentage of the total pages in the table that is to be appended to the end of the table as free space. For example, if *x* is 20, and the table has 100 data pages after the data has been loaded, 20 additional empty pages will be appended. The total number of data pages for the table will be 120. The data pages total does not factor in the number of index pages in the table. This option does not affect the index object. If two loads are done with this option specified, the second load will not reuse the extra space appended to the end by the first load.
usedefaults	If a source column for a target table column has been specified, but it contains no data for one or more row instances, default values are loaded. Examples of missing data are: • For DEL files: two adjacent column delimiters (",,") or two adjacent column delimiters separated by an arbitrary number of spaces (", ,") are specified for a column value. • For DEL/ASC/WSF files: A row that does not have enough columns, or is not long enough for the original specification. For ASC files, NULL column values are not considered explicitly missing, and a default will not be substituted for NULL column values. NULL column values are represented by all space characters for numeric, date, time, and /timestamp columns, or by using the NULL INDICATOR for a column of any type to indicate the column is NULL. Without this option, if a source column contains no data for a row instance, one of the following occurs: • For DEL/ASC/WSF files: If the column is nullable, a NULL is loaded. If the column is not nullable, the utility rejects the row.

Table 124. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL)

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the input data set. Converts character data (and numeric data specified in characters) from this code page to the database code page during the load operation. The following rules apply: • For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters. • `nullindchar` must specify symbols included in the standard ASCII set between code points x20 and x7F, inclusive. This refers to ASCII symbols and code points. EBCDIC data can use the corresponding symbols, even though the code points will be different. This option is not supported in conjunction with the CURSOR filetype.

Table 124. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

dateformat=*"x"*	*x* is the format of the date in the source file.[1] Valid date elements are:
	`YYYY - Year (four digits ranging from 0000 - 9999)`
	`M    - Month (one or two digits ranging from 1 - 12)`
	`MM   - Month (two digits ranging from 1 - 12; mutually exclusive with M)`
	`D    - Day (one or two digits ranging from 1 - 31)`
	`DD   - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD  - Day of the year (three digits ranging from 001 - 366; mutually exclusive with other day or month elements)`
	
	A default value of 1 is assigned for each element that is not specified. Some examples of date formats are:
	`"D-M-YYYY"`
	`"MM.DD.YYYY"`
	`"YYYYDDD"`

dumpfile = *x*	*x* is the fully qualified (according to the server database partition) name of an exception file to which rejected rows are written. A maximum of 32 KB of data is written per record. Following is an example that shows how to specify a dump file:
	`db2 load from data of del`
	`   modified by dumpfile = /u/user/filename`
	`   insert into table_name`
	
	The file will be created and owned by the instance owner. To override the default file permissions, use the `dumpfileaccessall` file type modifier.
	
	Notes:
	1. In a partitioned database environment, the path should be local to the loading database partition, so that concurrently running load operations do not attempt to write to the same file.
	2. The contents of the file are written to disk in an asynchronous buffered mode. In the event of a failed or an interrupted load operation, the number of records committed to disk cannot be known with certainty, and consistency cannot be guaranteed after a LOAD RESTART. The file can only be assumed to be complete for a load operation that starts and completes in a single pass.

dumpfileaccessall	Grants read access to 'OTHERS' when a dump file is created. This file type modifier is only valid when: 1. it is used in conjunction with `dumpfile` file type modifier 2. the user has SELECT privilege on the load target table 3. it is issued on a DB2 server database partition that resides on a UNIX operating system
fastparse	Reduced syntax checking is done on user-supplied column values, and performance is enhanced. Tables loaded under this option are guaranteed to be architecturally correct, and the utility is guaranteed to perform sufficient data checking to prevent a segmentation violation or trap. Data that is in correct form will be loaded correctly.
implieddecimal	The location of an implied decimal point is determined by the column definition; it is no longer assumed to be at the end of the value. For example, the value `12345` is loaded into a DECIMAL(8,2) column as `123.45`, *not* `12345.00`. This modifier cannot be used with the `packeddecimal` modifier.

Table 124. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timeformat="*x*"	*x* is the format of the time in the source file.[1] Valid time elements are:
	`H     - Hour (one or two digits ranging from 0 - 12`
	`            for a 12 hour system, and 0 - 24`
	`            for a 24 hour system)`
	`HH    - Hour (two digits ranging from 0 - 12`
	`            for a 12 hour system, and 0 - 24`
	`            for a 24 hour system; mutually exclusive`
	`            with H)`
	`M     - Minute (one or two digits ranging`
	`            from 0 - 59)`
	`MM    - Minute (two digits ranging from 0 - 59;`
	`            mutually exclusive with M)`
	`S     - Second (one or two digits ranging`
	`            from 0 - 59)`
	`SS    - Second (two digits ranging from 0 - 59;`
	`            mutually exclusive with S)`
	`SSSSS - Second of the day after midnight (5 digits`
	`            ranging from 00000 - 86399; mutually`
	`            exclusive with other time elements)`
	`TT    - Meridian indicator (AM or PM)`
	
	A default value of 0 is assigned for each element that is not specified. Some examples of time formats are:
	`"HH:MM:SS"`
	`"HH.MM TT"`
	`"SSSSS"`

Table 124. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat=″*x*″	*x* is the format of the time stamp in the source file.[1] Valid time stamp elements are:
	`YYYY   - Year (four digits ranging from 0000 - 9999)`
	`M      - Month (one or two digits ranging from 1 - 12)`
	`MM     - Month (two digits ranging from 01 - 12;`
	`            mutually exclusive with M and MMM)`
	`MMM    - Month (three-letter case-insensitive abbreviation for`
	`            the month name; mutually exclusive with M and MM)`
	`D      - Day (one or two digits ranging from 1 - 31)`
	`DD     - Day (two digits ranging from 1 - 31; mutually exclusive with D)`
	`DDD    - Day of the year (three digits ranging from 001 - 366;`
	`            mutually exclusive with other day or month elements)`
	`H      - Hour (one or two digits ranging from 0 - 12`
	`            for a 12 hour system, and 0 - 24 for a 24 hour system)`
	`HH     - Hour (two digits ranging from 0 - 12`
	`            for a 12 hour system, and 0 - 24 for a 24 hour system;`
	`            mutually exclusive with H)`
	`M      - Minute (one or two digits ranging from 0 - 59)`
	`MM     - Minute (two digits ranging from 0 - 59;`
	`            mutually exclusive with M, minute)`
	`S      - Second (one or two digits ranging from 0 - 59)`
	`SS     - Second (two digits ranging from 0 - 59;`
	`            mutually exclusive with S)`
	`SSSSS  - Second of the day after midnight (5 digits`
	`            ranging from 00000 - 86399; mutually`
	`            exclusive with other time elements)`
	`UUUUUU - Microsecond (6 digits ranging from 000000 - 999999;`
	`            mutually exclusive with all other microsecond elements)`
	`UUUUU  - Microsecond (5 digits ranging  from 00000 - 99999,`
	`            maps to range from 000000 - 999990;`
	`            mutually exclusive with all other microseond elements)`
	`UUUU   - Microsecond (4 digits ranging from 0000 - 9999,`
	`            maps to range from 000000 - 999900;`
	`            mutually exclusive with all other microseond elements)`
	`UUU    - Microsecond (3 digits ranging from 000 - 999,`
	`            maps to range from 000000 - 999000;`
	`            mutually exclusive with all other microseond elements)`
	`UU     - Microsecond (2 digits ranging from 00 - 99,`
	`            maps to range from 000000 - 990000;`
	`            mutually exclusive with all other microseond elements)`
	`U      - Microsecond (1 digit ranging from 0 - 9,`
	`            maps to range from 000000 - 900000;`
	`            mutually exclusive with all other microseond elements)`
	`TT     - Meridian indicator (AM or PM)`
	
	A default value of 1 is assigned for unspecified YYYY, M, MM, D, DD, or DDD elements. A default value of ′Jan′ is assigned to an unspecified MMM element. A default value of 0 is assigned for all other unspecified elements. Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	The valid values for the MMM element include: ′jan′, ′feb′, ′mar′, ′apr′, ′may′, ′jun′, ′jul′, ′aug′, ′sep′, ′oct′, ′nov′ and ′dec′. These values are case insensitive.
	
	The following example illustrates how to import data containing user defined date and time formats into a table called schedule:
	`db2 import from delfile2 of del`
	`   modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`   insert into schedule`

Table 124. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being loaded into graphic or double-byte character large object (DBCLOB) data field(s) is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic codepage is associated with the character code page. LOAD determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the database if the `codepage` modifier is not specified. This modifier should be used in conjunction with the delimited data file generated by drop table recovery only if the table being recovered has graphic data. **Restrictions** The `usegraphiccodepage` modifier MUST NOT be specified with DEL files created by the EXPORT utility, as these files contain data encoded in only one code page. The `usegraphiccodepage` modifier is also ignored by the double-byte character large objects (DBCLOBs) in files.

Table 125. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII)

Modifier	Description
binarynumerics	Numeric (but not DECIMAL) data must be in binary form, not the character representation. This avoids costly conversions. This option is supported only with positional ASC, using fixed length records specified by the `reclen` option. The following rules apply: • No conversion between data types is performed, with the exception of BIGINT, INTEGER, and SMALLINT. • Data lengths must match their target column definitions. • FLOATs must be in IEEE Floating Point format. • Binary data in the load source file is assumed to be big-endian, regardless of the platform on which the load operation is running. NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.
nullindchar=*x*	*x* is a single character. Changes the character denoting a NULL value to *x*. The default value of *x* is `Y`.[2] This modifier is case sensitive for EBCDIC data files, except when the character is an English letter. For example, if the NULL indicator character is specified to be the letter `N`, then `n` is also recognized as a NULL indicator.

Table 125. Valid file type modifiers for the load utility: ASC file formats (Non-delimited ASCII) (continued)

Modifier	Description

packeddecimal	Loads packed-decimal data directly, since the `binarynumerics` modifier does not include the DECIMAL field type.
	
	This option is supported only with positional ASC, using fixed length records specified by the `reclen` option.
	
	Supported values for the sign nibble are:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	NULLs cannot be present in the data for columns affected by this modifier. Blanks (normally interpreted as NULL) are interpreted as a binary value when this modifier is used.
	
	Regardless of the server platform, the byte order of binary data in the load source file is assumed to be big-endian; that is, when using this modifier on Windows operating systems, the byte order must not be reversed.
	
	This modifier cannot be used with the `implieddecimal` modifier.

reclen=*x*	*x* is an integer with a maximum value of 32 767. *x* characters are read for each row, and a new-line character is not used to indicate the end of the row.
striptblanks	Truncates any trailing blank spaces when loading data into a variable-length field. If this option is not specified, blank spaces are kept. This option cannot be specified together with `striptnulls`. These are mutually exclusive options. This option replaces the obsolete `t` option, which is supported for back-level compatibility only.
striptnulls	Truncates any trailing NULLs (0x00 characters) when loading data into a variable-length field. If this option is not specified, NULLs are kept. This option cannot be specified together with `striptblanks`. These are mutually exclusive options. This option replaces the obsolete `padwithzero` option, which is supported for back-level compatibility only.

zoneddecimal	Loads zoned decimal data, since the BINARYNUMERICS modifier does not include the DECIMAL field type. This option is supported only with positional ASC, using fixed length records specified by the RECLEN option.
	
	Half-byte sign values can be one of the following:
	`+ = 0xC 0xA 0xE 0xF`
	`- = 0xD 0xB`
	
	Supported values for digits are `0x0` to `0x9.`
	
	Supported values for zones are `0x3` and `0xF.`

File type modifiers for the load utility

Table 126. Valid file type modifiers for the load utility: DEL file formats (Delimited ASCII)

Modifier	Description
chardel*x*	*x* is a single character string delimiter. The default value is a double quotation mark ("). The specified character is used in place of double quotation marks to enclose a character string.[23] If you wish to explicitly specify the double quotation mark(") as the character string delimiter, you should specify it as follows: `modified by chardel""` The single quotation mark (') can also be specified as a character string delimiter as follows: `modified by chardel''`
coldel*x*	*x* is a single character column delimiter. The default value is a comma (,). The specified character is used in place of a comma to signal the end of a column.[23]
decplusblank	Plus sign character. Causes positive decimal values to be prefixed with a blank space instead of a plus sign (+). The default action is to prefix positive decimal values with a plus sign.
decpt*x*	*x* is a single character substitute for the period as a decimal point character. The default value is a period (.). The specified character is used in place of a period as a decimal point character.[23]
delprioritychar	The current default priority for delimiters is: record delimiter, character delimiter, column delimiter. This modifier protects existing applications that depend on the older priority by reverting the delimiter priorities to: character delimiter, record delimiter, column delimiter. Syntax: `db2 load ... modified by delprioritychar ...` For example, given the following DEL data file: `"Smith, Joshua",4000,34.98<row delimiter>` `"Vincent,<row delimiter>, is a manager", ...` `... 4005,44.37<row delimiter>` With the `delprioritychar` modifier specified, there will be only two rows in this data file. The second <row delimiter> will be interpreted as part of the first data column of the second row, while the first and the third <row delimiter> are interpreted as actual record delimiters. If this modifier is *not* specified, there will be three rows in this data file, each delimited by a <row delimiter>.
keepblanks	Preserves the leading and trailing blanks in each field of type CHAR, VARCHAR, LONG VARCHAR, or CLOB. Without this option, all leading and tailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields. The following example illustrates how to load data into a table called TABLE1, while preserving all leading and trailing spaces in the data file: `db2 load from delfile3 of del` `modified by keepblanks` `insert into table1`
nochardel	The load utility will assume all bytes found between the column delimiters to be part of the column's data. Character delimiters will be parsed as part of column data. This option should not be specified if the data was exported using DB2 (unless nochardel was specified at export time). It is provided to support vendor data files that do not have character delimiters. Improper usage might result in data loss or corruption. This option cannot be specified with `chardelx`, `delprioritychar` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.

Table 127. Valid file type modifiers for the load utility: IXF file format

Modifier	Description
forcein	Directs the utility to accept data despite code page mismatches, and to suppress translation between code pages. Fixed length target fields are checked to verify that they are large enough for the data. If `nochecklengths` is specified, no checking is done, and an attempt is made to load each row.
nochecklengths	If `nochecklengths` is specified, an attempt is made to load each row, even if the source data has a column definition that exceeds the size of the target table column. Such rows can be successfully loaded if code page conversion causes the source data to shrink; for example, 4-byte EUC data in the source could shrink to 2-byte DBCS data in the target, and require half the space. This option is particularly useful if it is known that the source data will fit in all cases despite mismatched column definitions.

Notes:

1. Double quotation marks around the date format string are mandatory. Field separators cannot contain any of the following: a-z, A-Z, and 0-9. The field separator should not be the same as the character delimiter or field delimiter in the DEL file format. A field separator is optional if the start and end positions of an element are unambiguous. Ambiguity can exist if (depending on the modifier) elements such as D, H, M, or S are used, because of the variable length of the entries.

   For time stamp formats, care must be taken to avoid ambiguity between the month and the minute descriptors, since they both use the letter M. A month field must be adjacent to other date fields. A minute field must be adjacent to other time fields. Following are some ambiguous time stamp formats:

   ```
   "M" (could be a month, or a minute)
   "M:M" (Which is which?)
   "M:YYYY:M" (Both are interpreted as month.)
   "S:M:YYYY" (adjacent to both a time value and a date value)
   ```

   In ambiguous cases, the utility will report an error message, and the operation will fail.

   Following are some unambiguous time stamp formats:

   ```
   "M:YYYY" (Month)
   "S:M" (Minute)
   "M:YYYY:S:M" (Month....Minute)
   "M:H:YYYY:M:D" (Minute....Month)
   ```

   Some characters, such as double quotation marks and back slashes, must be preceded by an escape character (for example, \).
2. The character must be specified in the code page of the source data.

   The character code point (instead of the character symbol), can be specified using the syntax xJJ or 0xJJ, where JJ is the hexadecimal representation of the code point. For example, to specify the # character as a column delimiter, use one of the following:

   ```
   ... modified by coldel# ...
   ... modified by coldel0x23 ...
   ... modified by coldelX23 ...
   ```
3. Delimiter restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.

4. The load utility does not issue a warning if an attempt is made to use unsupported file types with the MODIFIED BY option. If this is attempted, the load operation fails, and an error code is returned.

Table 128. LOAD behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	LOAD behavior
Absent	Absent	All data in the file is assumed to be in the database code page, not the application code page, even if the CLIENT option is specified.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the database code page, even if the CLIENT option is specified. Graphic data is assumed to be in the code page of the database graphic data, even if the CLIENT option is specified. If the database code page is single-byte, then all data is assumed to be in the database code page. **Warning:** Graphic data will be corrupted when loaded into a single-byte database.
Present	Present	Character data is assumed to be in code page `N`. Graphic data is assumed to be in the graphic code page of `N`. If `N` is a single-byte or double-byte code page, then all data is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when loaded into the database if `N` is a single-byte code page.

Related reference:

- "db2Load - Load data into a table" on page 723
- "Delimiter restrictions for moving data" in *Command Reference*
- "LOAD " on page 542

MIGRATE DATABASE

Converts previous versions of DB2 databases to the formats corresponding to the release run by the instance.

The `db2ckmig` command must be issued prior to migrating the intance. The `db2imigr` command implicitly calls the `db2ckmig`. Backup all databases prior to migration, and prior to the installation of the current version of DB2 database product on Windows operating systems.

Authorization:

sysadm

Required connection:

This command establishes a database connection.

Command syntax:

```
>>──MIGRATE──┬─DATABASE─┬──database-alias──────────────────────────>
             └─DB───────┘

>──┬───────────────────────────────────┬──────────────────────────><
   └─USER──username──┬───────────────┬─┘
                     └─USING──password─┘
```

Command parameters:

DATABASE database-alias
: Specifies the alias of the database to be migrated to the currently installed version of the database manager.

USER username
: Identifies the user name under which the database is to be migrated.

USING password
: The password used to authenticate the user name. If the password is omitted, but a user name was specified, the user is prompted to enter it.

Examples:

The following example migrates the database cataloged under the database alias `sales`:

```
db2 migrate database sales
```

Usage notes:

This command will only migrate a database to a newer version, and cannot be used to convert a migrated database to its previous version.

The database must be cataloged before migration.

If an error occurs during migration, it might be necessary to issue the TERMINATE command before attempting the suggested user response. For example, if a log full error occurs during migration (SQL1704: Database migration failed. Reason code ″3″.), it will be necessary to issue the TERMINATE command before increasing the values of the database configuration parameters LOGPRIMARY and LOGFILSIZ. The CLP must refresh its database directory cache if the migration failure occurs after the database has already been relocated (which is likely to be the case when a ″log full″ error returns).

Related tasks:

- "Migrating databases" in *Migration Guide*

Related reference:

- "TERMINATE command" in *Command Reference*
- "sqlemgdb - Migrate previous version of DB2 database to current version" on page 815

QUIESCE

Scope:

QUIESCE DATABASE results in all objects in the database being in the quiesced mode. Only the allowed user/group and *sysadm*, *sysmaint*, *dbadm*, or *sysctrl* will be able to access the database or its objects.

Authorization:

One of the following:

For database level quiesce:
- *sysadm*
- *dbadm*
- *sysadm*
- *sysctrl*

Required connection:

Database

Command syntax:

```
►►──QUIESCE──┬─DATABASE─┬──┬─IMMEDIATE──────────────────────┬──────────────────►
             └─DB───────┘  └─DEFER──┬─────────────────────┬─┘
                                    └─WITH TIMEOUT──minutes─┘

  ┌─FORCE CONNECTIONS─┐
►─┴───────────────────┴─────────────────────────────────────────────────────►◄
```

Command parameters:

DEFER
: Wait for applications until they commit the current unit of work.

  WITH TIMEOUT
  : Specifies a time, in minutes, to wait for applications to commit the current unit of work. If no value is specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the *start_stop_timeout* database manager configuration parameter will be used.

IMMEDIATE
: Do not wait for the transactions to be committed, immediately rollback the transactions.

FORCE CONNECTIONS
: Force the connections off.

DATABASE
: Quiesce the database. All objects in the database will be placed in quiesced mode. Only specified users in specified groups and users with *sysadm*, *sysmaint*, and *sysctrl* authority will be able to access to the database or its objects.

Usage notes:

- After QUIESCE DATABASE, users with *sysadm*, *sysmaint*, *sysctrl*, or *dbadm* authority, and GRANT/REVOKE privileges can designate who will be able to connect. This information will be stored permanently in the database catalog tables.

  For example,

  ```
  grant quiesce_connect on database to <username/groupname>
  revoke quiesce_connect on database from <username/groupname>
  ```

Related reference:

- "UNQUIESCE " on page 626
- "db2DatabaseQuiesce - Quiesce the database" on page 681
- "db2InstanceQuiesce - Quiesce instance" on page 712
- "QUIESCE DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

QUIESCE TABLESPACES FOR TABLE

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive. There are three possible states resulting from the quiesce function:

- Quiesced: SHARE
- Quiesced: UPDATE
- Quiesced: EXCLUSIVE

Scope:

In a single-partition environment, this command quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load operation. In a partitioned database environment, this command acts locally on a database partition. It quiesces only that portion of table spaces belonging to the database partition on which the load operation is performed. For partitioned tables, all of the table spaces listed in SYSDATAPARTITIONS.TBSPACEID and SYSDATAPARTITIONS.LONG_TBSPACEID associated with a table and with a status of normal, attached or detached, (for example, SYSDATAPARTITIONS.STATUS of ’"’, ’A’ or ’D’, respectively) are quiesced.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- *load*

Required connection:

Database

Command syntax:

QUIESCE TABLESPACES FOR TABLE

```
►►──QUIESCE TABLESPACES FOR TABLE──┬─tablename────────┬──┬─SHARE───────────┬──────►◄
                                   └─schema.tablename─┘  ├─INTENT TO UPDATE─┤
                                                         ├─EXCLUSIVE────────┤
                                                         └─RESET────────────┘
```

Command parameters:

TABLE

tablename
: Specifies the unqualified table name. The table cannot be a system catalog table.

schema.tablename
: Specifies the qualified table name. If *schema* is not provided, the CURRENT SCHEMA will be used. The table cannot be a system catalog table.

SHARE
: Specifies that the quiesce is to be in share mode.

When a ″quiesce share″ request is made, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table, so that the state is persistent. The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces are allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

INTENT TO UPDATE
: Specifies that the quiesce is to be in intent to update mode.

When a ″quiesce intent to update″ request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces is recorded in the table space table.

EXCLUSIVE
: Specifies that the quiesce is to be in exclusive mode.

When a ″quiesce exclusive″ request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer) has exclusive access to the table and the table spaces.

RESET
: Specifies that the state of the table spaces is to be reset to normal. A quiesce state cannot be reset if the connection that issued the quiesce request is still active.

Example:

Usage notes:

This command is not supported for declared temporary tables.

A quiesce is a persistent lock. Its benefit is that it persists across transaction failures, connection failures, and even across system failures (such as power failure, or reboot).

A quiesce is owned by a connection. If the connection is lost, the quiesce remains, but it has no owner, and is called a *phantom quiesce*. For example, if a power outage caused a load operation to be interrupted during the delete phase, the table spaces for the loaded table would be left in delete pending, quiesce exclusive state. Upon database restart, this quiesce would be an unowned (or phantom) quiesce. The removal of a phantom quiesce requires a connection with the same user ID used when the quiesce mode was set.

To remove a phantom quiesce:

1. Connect to the database with the same user ID used when the quiesce mode was set.
2. Use the LIST TABLESPACES command to determine which table space is quiesced.
3. Re-quiesce the table space using the current quiesce state. For example:

Once completed, the new connection owns the quiesce, and the load operation can be restarted.

There is a limit of five quiescers on a table space at any given time.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

Related reference:

- “sqluvqdp - Quiesce table spaces for a table” on page 827
- “LIST TABLESPACES ” on page 537
- “LOAD ” on page 542
- “QUIESCE ” on page 567
- “UNQUIESCE ” on page 626
- “QUIESCE TABLESPACES FOR TABLE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

RECOVER DATABASE

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope:

In a partitioned database environment, this command can only be invoked from the catalog partition. A database recover operation to a specified point in time affects all database partitions that are listed in the `db2nodes.cfg` file. A database recover operation to the end of logs affects the database partitions that are specified. If no partitions are specified, it affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization:

To recover an existing database, one of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

To recover to a new database, one of the following:
- *sysadm*
- *sysctrl*

Required connection:

To recover an existing database, a database connection is required. This command automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. To recover to a new database, an instance attachment and a database connection are required. The instance attachment is required to create the database.

Command syntax:

```
►►──RECOVER──┬─DATABASE─┬──source-database-alias──────────────────────────►
             └─DB───────┘

►──┬──────────────────────────────────────────────────────────────┬────────►
   │                  ┌─USING LOCAL TIME─┐                        │
   └─TO──┬─isotime──┼──────────────────┼──┬─────────────────────┬┘
         │          └─USING UTC TIME───┘  └─ON ALL DBPARTITIONNUMS─┘│
         └─END OF LOGS──┬─────────────────────────────────────────┘
                        └─┤ On Database Partition clause ├─┘

►──┬─────────────────────────────────────┬─────────────────────────────────►
   └─USER──username──┬─────────────────┬─┘
                     └─USING──password─┘

►──┬──────────────────────────────────────────────────────────────────┬───►
   └─USING HISTORY FILE──(──history-file──┬─────────────────────────┬─)─┘
                                          └─,─┤ History File clause ├─┘

►──┬──────────────────────────────────────────────────────────────────┬───►
   └─OVERFLOW LOG PATH──(──log-directory──┬─────────────────────────┬─)─┘
                                          └─,─┤ Log Overflow clause ├─┘

►──┬──────────────────┬──┬───────────────────────────┬──┬─────────┬───────►◄
   └─COMPRLIB──lib-name─┘  └─COMPROPTS──options-string─┘  └─RESTART─┘
```

On Database Partition clause:

```
├──ON──┬─┤ Database Partition List clause ├───────────────────────────┬──────┤
       └─ALL DBPARTITIONNUMS──┬──────────────────────────────────────┬─┘
                              └─EXCEPT──┤ Database Partition List clause ├─┘
```

Database Partition List clause:

```
                                    ┌─,───────────────────────────────────────────────┐
                                    ▼                                                  │
├──┬─DBPARTITIONNUM──┬──(────db-partition-number1──┬───────────────────────────┬─┴──)──────────┤
   └─DBPARTITIONNUMS─┘                             └─TO──db-partition-number2──┘
```

Log Overflow clause:

```
   ┌─,──────────────────────────────────────────────────────────┐
   ▼                                                            │
├────log-directory──ON DBPARTITIONNUM──db-partition-number1──┴──────────────────┤
```

History File clause:

```
   ┌─,──────────────────────────────────────────────────────────┐
   ▼                                                            │
├────history-file──ON DBPARTITIONNUM──db-partition-number1──┴───────────────────┤
```

Command parameters:

DATABASE *database-alias*
: The alias of the database that is to be recovered.

USER *username*
: The user name under which the database is to be recovered.

USING *password*
: The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

TO

isotime The point in time to which all committed transactions are to be recovered (including the transaction committed precisely at that time, as well as all transactions committed previously).

This value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year, month, day, hour, minutes, seconds, microseconds), expressed in Coordinated Universal Time (UTC, formerly known as GMT). UTC helps to avoid having the same time stamp associated with different logs (because of a change in time associated with daylight savings time, for example). The time stamp in a backup image is based on the local time at which the backup operation started. The CURRENT TIMEZONE special register specifies the difference between UTC and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits represent the number of hours, the next two digits represent the number of minutes, and the last two digits represent the number of seconds). Subtracting CURRENT TIMEZONE from a local time converts that local time to UTC.

USING LOCAL TIME
: Specifies the point in time to which to recover. This option allows the user to recover to a point in time that is the server's local time rather than UTC time.

Notes:

1. If the user specifies a local time for recovery, all messages returned to the user will also be in local time. All times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. The timestamp string is converted to UTC on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client.
3. If the timestamp string is close to the time change of the clock due to daylight saving time, it is important to know if the stop time is before or after the clock change, and specify it correctly.

USING UTC TIME
Specifies the point in time to which to recover.

END OF LOGS
Specifies that all committed transactions from all online archive log files listed in the database configuration parameter *logpath* are to be applied.

ON ALL DBPARTITIONNUMS
Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file. This is the default if a database partition clause is not specified.

EXCEPT
Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

ON DBPARTITIONNUM / ON DBPARTITIONNUMS
Roll the database forward on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

USING HISTORY FILE *history-file*

history-file **ON DBPARTITIONNUM**
In a partitioned database environment, allows a different history file

OVERFLOW LOG PATH *log-directory*
Specifies an alternate log path to be searched for archived logs during recovery. Use this parameter if log files were moved to a location other than that specified by the *logpath* database configuration parameter. In a partitioned database environment, this is the (fully qualified) default overflow log path *for all database partitions*. A relative overflow log path can be specified for single-partition databases.

The OVERFLOW LOG PATH command parameter will overwrite the value (if any) of the database configuration parameter *overflowlogpath*.

COMPRLIB *lib-name*
Indicates the name of the library to be used to perform the decompression.

The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *options-string*
: Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is "@", the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

RESTART
: The `RESTART` keyword can be used if a prior RECOVER operation was interrupted or otherwise did not complete. Starting in v91, a subsequent `RECOVER command` will attempt to continue the previous `RECOVER`, if possible. Using the `RESTART` keyword forces `RECOVER` to start with a fresh restore and then rollforward to the PIT specified.

log-directory **ON DBPARTITIONNUM**
: In a partitioned database environment, allows a different log path to override the default overflow log path for a specific database partition.

Examples:

In a single-partition database environment, where the database being recovered currently exists, and so the most recent version of the history file is available in the *dftdbpath*:

1. To use the latest backup image and rollforward to the end of logs using all default values:
   ```
   RECOVER DB SAMPLE
   ```
2. To recover the database to a PIT, issue the following. The most recent image that can be used will be restored, and logs applied until the PIT is reached.
   ```
   RECOVER DB SAMPLE TO  2001-12-31-04.00.00
   ```
3. To recover the database using a saved version of the history file. issue the following. For example, if the user needs to recover to an extremely old PIT which is no longer contained in the current history file, the user will have to provide a version of the history file from this time period. If the user has saved a history file from this time period, this version can be used to drive the recover.
   ```
   RECOVER DB SAMPLE TO  1999-12-31-04.00.00
      USING HISTORY FILE (/home/user/old1999files/db2rhist.asc)
   ```

In a single-partition database environment, where the database being recovered does not exist, you must use the USING HISTORY FILE clause to point to a history file.

1. If you have not made any backups of the history file, so that the only version available is the copy in the backup image, the recommendation is to issue a RESTORE followed by a ROLLFORWARD. However, to use RECOVER, you would first have to extract the history file from the image to some location, for example /home/user/oldfiles/db2rhist.asc, and then issue this command. (This

version of the history file does not contain any information about log files that are required for rollforward, so this history file is not useful for RECOVER.)

```
RECOVER DB SAMPLE TO END OF LOGS
   USING HISTORY FILE (/home/user/fromimage/db2rhist.asc)
```

2. If you have been making periodic or frequent backup copies of the history, the USING HISTORY clause should be used to point to this version of the history file. If the file is /home/user/myfiles/db2rhist.asc, issue the command:

   ```
   RECOVER DB SAMPLE TO PIT
      USING HISTORY FILE (/home/user/myfiles/db2rhist.asc)
   ```

   (In this case, you can use any copy of the history file, not necessarily the latest, as long as it contains a backup taken before the point-in-time (PIT) requested.)

In a partitioned database envrionment, where the database exists on all database partitions, and the latest history file is available on *dftdbpath* on all database partitions:

1. To recover the database to a PIT on all nodes. DB2 will verify that the PIT is reachable on all nodes before starting any restore operations.

   ```
   RECOVER DB SAMPLE TO  2001-12-31-04.00.00
   ```

2. To recover the database to this PIT on all nodes. DB2 will verify that the PIT is reachable on all nodes before starting any restore operations. The RECOVER operation on each node is identical to a single-partition RECOVER.

   ```
   RECOVER DB SAMPLE TO END OF LOGS
   ```

3. Even though the most recent version of the history file is in the *dftdbpath*, you might want to use several specific history files. Unless otherwise specified, each database partition will use the history file found locally at /home/user/oldfiles/db2rhist.asc. The exceptions are nodes 2 and 4. Node 2 will use: /home/user/node2files/db2rhist.asc, and node 4 will use: /home/user/node4files/db2rhist.asc.

   ```
   RECOVER DB SAMPLE TO  1999-12-31-04.00.00
      USING HISTORY FILE (/home/user/oldfiles/db2rhist.asc,
         /home/user/node2files/db2rhist.asc ON DBPARTITIONNUM 2,
         /home/user/node4files/db2rhist.asc ON DBPARTITIONNUM 4)
   ```

4. It is possible to recover a subset of nodes instead of all nodes, however a PIT RECOVER can not be done in this case, the recover must be done to EOL.

   ```
   RECOVER DB SAMPLE TO END OF LOGS ON DBPARTITIONNUMS(2 TO 4, 7, 9)
   ```

In a partitioned database environment, where the database does not exist:

1. If you have not made any backups of the history file, so that the only version available is the copy in the backup image, the recommendation is to issue a RESTORE followed by a ROLLFORWARD. However, to use RECOVER, you would first have to extract the history file from the image to some location, for example, /home/user/oldfiles/db2rhist.asc, and then issue this command. (This version of the history file does not contain any information about log files that are required for rollforward, so this history file is not useful for the recover.)

   ```
   RECOVER DB SAMPLE TO PIT
      USING HISTORY FILE (/home/user/fromimage/db2rhist.asc)
   ```

2. If you have been making periodic or frequent backup copies of the history, the USING HISTORY clause should be used to point to this version of the history file. If the file is /home/user/myfiles/db2rhist.asc, you can issue the following command:

   ```
   RECOVER DB SAMPLE TO END OF LOGS
      USING HISTORY FILE (/home/user/myfiles/db2rhist.asc)
   ```

Usage notes:

- Recovering a database might require a load recovery using tape devices. If prompted for another tape, the user can respond with one of the following:

c	Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted).
d	Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes).
t	Terminate. Terminate all devices.

- If there is a failure during the restore portion of the recover operation, you can reissue the RECOVER DATABASE command. If the restore operation was successful, but there was an error during the rollforward operation, you can issue a ROLLFORWARD DATABASE command, since it is not necessary (and it is time-consuming) to redo the entire recover operation.
- In a partitioned database environment, if there is an error during the restore portion of the recover operation, it is possible that it is only an error on a single database partition. Instead of reissuing the RECOVER DATABASE command, which restores the database on all database partitions, it is more efficient to issue a RESTORE DATABASE command for the database partition that failed, followed by a ROLLFORWARD DATABASE command.

Related concepts:

- "Developing a backup and recovery strategy" in *Data Recovery and High Availability Guide and Reference*

Related tasks:

- "Using recover" in *Data Recovery and High Availability Guide and Reference*

REDISTRIBUTE DATABASE PARTITION GROUP

Redistributes data across the database partitions in a database partition group. The current data distribution, whether it is uniform or skewed, can be specified. The redistribution algorithm selects the partitions to be moved based on the current data distribution.

Scope:

This command affects all database partitions in the database partition group.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*(DELETE, INSERT, and UPDATE authority is also required on all tables in the database partition group being redistributed)
- *dbadm*

Command syntax:

```
►►──REDISTRIBUTE DATABASE PARTITION GROUP──database partition group──────────────►
```

```
►──┬──┬─UNIFORM──────────────────┬──┬──────────────────────────────────────────►◄
   │  └─USING DISTFILE──distfile─┘  │
   ├─USING TARGETMAP──targetmap─────┤
   ├─CONTINUE───────────────────────┤
   └─ROLLBACK───────────────────────┘
```

Command parameters:

DATABASE PARTITION GROUP database partition group
: The name of the database partition group. This one-part name identifies a database partition group described in the SYSCAT.DBPARTITIONGROUPS catalog table. The database partition group cannot currently be undergoing redistribution. Tables in the IBMCATGROUP and the IBMTEMPGROUP database partition groups cannot be redistributed.

UNIFORM
: Specifies that the data is uniformly distributed across hash partitions (that is, every hash partition is assumed to have the same number of rows), but the same number of hash partitions do not map to each database partition. After redistribution, all database partitions in the database partition group have approximately the same number of hash partitions.

USING DISTFILE distfile
: If the distribution of distribution key values is skewed, use this option to achieve a uniform redistribution of data across the database partitions of a database partition group.

Use the *distfile* to indicate the current distribution of data across the 4 096 hash partitions.

Use row counts, byte volumes, or any other measure to indicate the amount of data represented by each hash partition. The utility reads the integer value associated with a partition as the weight of that partition. When a *distfile* is specified, the utility generates a target distribution map that it uses to redistribute the data across the database partitions in the database partition group as uniformly as possible. After the redistribution, the weight of each database partition in the database partition group is approximately the same (the weight of a database partition is the sum of the weights of all database partitions that map to that database partition).

For example, the input distribution file might contain entries as follows:

```
10223
1345
112000
0
100
...
```

In the example, hash partition 2 has a weight of 112 000, and partition 3 (with a weight of 0) has no data mapping to it at all.

The *distfile* should contain 4 096 positive integer values in character format. The sum of the values should be less than or equal to 4 294 967 295.

USING TARGETMAP targetmap
: The file specified in *targetmap* is used as the target distribution map. Data redistribution is done according to this file.

If a database partition included in the target map is not in the database partition group, an error is returned. Issue ALTER DATABASE PARTITION GROUP ADD DBPARTITIONNUM before running REDISTRIBUTE DATABASE PARTITION GROUP.

If a database partition excluded from the target map *is* in the database partition group, that database partition will not be included in the partitioning. Such a database partition can be dropped using ALTER DATABASE PARTITION GROUP DROP DBPARTITIONNUM either before or after REDISTRIBUTE DATABASE PARTITION GROUP.

CONTINUE
Continues a previously failed REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

ROLLBACK
Rolls back a previously failed REDISTRIBUTE DATABASE PARTITION GROUP operation. If none occurred, an error is returned.

Usage notes:

- When a redistribution operation is done, a message file is written to:
  - The `/sqllib/redist` directory on UNIX based systems, using the following format for subdirectories and file name: *database-name.database-partition-group-name.timestamp*.
  - The `\sqllib\redist\` directory on Windows operating systems, using the following format for subdirectories and file name: *database-name\first-eight-characters-of-the-database-partition-group-name\date\time*.
- The time stamp value is the time when the command was issued.
- This utility performs intermittent COMMITs during processing.
- Use the ALTER DATABASE PARTITION GROUP statement to add database partitions to a database partition group. This statement permits one to define the containers for the table spaces associated with the database partition group.
- DB2 Parallel Edition for AIX Version 1 syntax, with ADD DBPARTITIONNUM and DROP DBPARITITIONNUM options, is supported for users with *sysadm* or *sysctrl* authority. For ADD DBPARTITIONNUM, containers are created like the containers on the lowest node number of the existing nodes within the database partition group.
- All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.
- It is also recommended to update statistics by issuing RUNSTATS after the redistribute database partition group operation has completed.
- Database partition groups containing replicated summary tables or tables defined with DATA CAPTURE CHANGES cannot be redistributed.
- Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables in the database partition group.
- Before starting a redistribute operation, ensure there are no tables in the Load Pending state. Table states can be checked by using the `LOAD QUERY` command. If you discover data in the wrong database partition as a result of a redistribute operation, there are two options. You can:
  1. unload the table, drop it and then reload the table, or

2. use a new target map to redistribute the database partition group again.

Compatibilities:

For compatibility with versions earlier than Version 8:

- The keyword NODEGROUP can be substituted for DATABASE PARTITION GROUP.

Related tasks:

- "Redistributing data across database partitions" on page 339
- Chapter 26, "Altering a database partition group," on page 325

Related reference:

- "sqludrdt - Redistribute data across a database partition group" on page 820
- "LIST DATABASE DIRECTORY command" in *Command Reference*
- "RUNSTATS command" in *Command Reference*
- "REBIND " on page 659
- "REDISTRIBUTE DATABASE PARTITION GROUP command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

REORG INDEXES/TABLE

Reorganizes an index or a table.

You can reorganize all indexes defined on a table by rebuilding the index data into unfragmented, physically contiguous pages. Alternatively, you have the option of reorganizing specific indexes on a range partitioned table.

If you specify the CLEANUP ONLY option of the index clause, cleanup is performed without rebuilding the indexes. This command cannot be used against indexes on declared temporary tables (SQLSTATE 42995).

The table option reorganizes a table by reconstructing the rows to eliminate fragmented data, and by compacting information.

Scope:

This command affects all database partitions in the database partition group.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*
- CONTROL privilege on the table.

Required connection:

Database

Command syntax:

```
►►──REORG──────────────────────────────────────────────────────────────────────►

►──┬─TABLE──table-name──┤ Table Clause ├──────────────────────────────────────┬──►
   └─┬─INDEXES ALL FOR TABLE──table-name──────────────┬─┤ Index Clause ├──────┘
     └─INDEX──index-name──┬──────────────────────────┤
                          └─FOR TABLE──table-name─────┘

►──┬────────────────────────────────┬──────────────────────────────────────────►◄
   └─┤ Database Partition Clause ├──┘
```

Table Clause:

```
├──┬─────────────────────┬──────────────────────────────────────────────────────►
   └─INDEX──index-name───┘

                                                                      ┌─KEEPDICTIONARY──┐
►──┬─┬────────────────────┬──┬────────────────────┬──┬───────────┬──┬─LONGLOBDATA──┬─────────────────────────┬─┬──┼─────────────────┼─┬──┤
   │ ├─ALLOW NO ACCESS────┤  └─USE──tbspace-name──┘  └─INDEXSCAN─┘                 └─USE──longtbspace-name───┘    └─RESETDICTIONARY─┘ │
   │ └─ALLOW READ ACCESS──┘                                                                                                                │
   │               ┌─ALLOW WRITE ACCESS─┐                      ┌─START──┐                                                              │
   └─INPLACE──┬─┬──┴─────────────────────┴──┬──────────────────┬─┴────────┴─┬─┬───────────────────────────────────────────────────────┘
              │ └─ALLOW READ ACCESS──┘      └─NOTRUNCATE TABLE─┘ └─RESUME─┘ │
              ├─STOP──────────────────────────────────────────────────────┤
              └─PAUSE─────────────────────────────────────────────────────┘
```

Index Clause:

```
├──┬──────────────────────┬──┬────────────────────────────────────┬──────────┤
   ├─ALLOW NO ACCESS──────┤  │                  ┌─ALL────┐         │
   ├─ALLOW WRITE ACCESS───┤  ├─CLEANUP ONLY─────┼────────┼─────────┤
   └─ALLOW READ ACCESS────┘  │                  └─PAGES──┘         │
                             └─CONVERT─────────────────────────────┘
```

Database Partition Clause:

```
                                       ┌─,─────────────────────────────────────────────┐
                                       ▼                                               │
├──ON──┬─┬─DBPARTITIONNUM──┬──(──────db-partition-number1──┬──────────────────────────┬─┴──)──┬──────┤
       │ └─DBPARTITIONNUMS─┘                               └─TO──db-partition-number2─┘      │
       └─ALL DBPARTITIONNUMS──┬──────────────────────────────────────────────────────────────┤
                              │                                     ┌─,──────────────────────────────────────────────┐
                              │                                     ▼                                                │
                              └─EXCEPT──┬─DBPARTITIONNUM──┬──(──────db-partition-number1──┬──────────────────────────┬─┴──)─┘
                                        └─DBPARTITIONNUMS─┘                               └─TO──db-partition-number2─┘
```

Command parameters:

INDEXES ALL FOR TABLE table-name
: Specifies the table whose indexes are to be reorganized. The table can be in a local or a remote database.

INDEX index-name
: Specifies an individual index to be reorganized on a partitioned table. Reorganization of individual indexes are ONLY supported for non-partitioned indexes on a partitioned table. This parameter is not supported for block indexes.

FOR TABLE table-name
: Specifies the table name location of the individual index being reorganized on a partitioned table. This parameter is optional, given that index names are unique across the database.

ALLOW NO ACCESS
Specifies that no other users can access the table while the indexes are being reorganized.

ALLOW READ ACCESS
Specifies that other users can have read-only access to the table while the indexes are being reorganized. This access level is not supported for REORG INDEXES of a partitioned table unless the CLEANUP ONLY option is specified.

ALLOW WRITE ACCESS
Specifies that other users can read from and write to the table while the indexes are being reorganized. This access level is not supported for multi-dimensionally clustered (MDC) tables, partitioned tables, extended indexes, or tables containing a column with the XML data type unless the `CLEANUP ONLY` option is specified.

When no ACCESS mode is specified, one will be chosen for you in the following way:

Table 129. Default table access chosen based on the command, table type and additional parameters specified for the index clause:

Command	Table type	Additional parameters specified for index clause	Default access mode
REORG INDEXES	non-partitioned table	any	ALLOW READ ACCESS
REORG INDEXES	partitioned table	none specified	ALLOW NO ACCESS
REORG INDEXES	partitioned table	CLEANUP ONLY specified	ALLOW READ ACCESS
REORG INDEX	partitioned table	any	ALLOW READ ACCESS

CLEANUP ONLY
When CLEANUP ONLY is requested, a cleanup rather than a full reorganization will be done. The indexes will not be rebuilt and any pages freed up will be available for reuse by indexes defined on this table only.

The CLEANUP ONLY PAGES option will search for and free committed pseudo empty pages. A committed pseudo empty page is one where all the keys on the page are marked as deleted and all these deletions are known to be committed. The number of pseudo empty pages in an indexes can be determined by running runstats and looking at the NUM EMPTY LEAFS column in SYSCAT.INDEXES. The PAGES option will clean the NUM EMPTY LEAFS if they are determined to be committed.

The CLEANUP ONLY ALL option will free committed pseudo empty pages, as well as remove committed pseudo deleted keys from pages that are not pseudo empty. This option will also try to merge adjacent leaf pages if doing so will result in a merged leaf page that has at least PCTFREE free space on the merged leaf page, where PCTFREE is the percent free space defined for the index at index creation time. The default PCTFREE is ten percent. If two pages can be merged, one of the pages will be freed. The number of pseudo deleted keys in an index , excluding those on pseudo empty pages, can be determined by running runstats and then selecting the NUMRIDS DELETED from SYSCAT.INDEXES. The ALL option will clean the NUMRIDS DELETED and the NUM EMPTY LEAFS if they are determined to be committed.

ALL Specifies that indexes should be cleaned up by removing committed pseudo deleted keys and committed pseudo empty pages.

PAGES
Specifies that committed pseudo empty pages should be removed from the index tree. This will not clean up pseudo deleted keys on pages that are not pseudo empty. Since it is only checking the pseudo empty leaf pages, it is considerably faster than using the ALL option in most cases.

CONVERT
If you are not sure whether the table you are operating on has a type-1 or type-2 index, but want type-2 indexes, you can use the CONVERT option. If the index is type 1, this option will convert it into type 2. If the index is already type 2, this option has no effect.

All indexes created by DB2 prior to Version 8 are type-1 indexes. All indexes created by Version 8 are Type 2 indexes, except when you create an index on a table that already has a type 1 index. In this case the new index will also be of type 1.

Using the INSPECT command to determine the index type can be slow. CONVERT allows you to ensure that the new index will be Type 2 without your needing to determine its original type.

Use the ALLOW READ ACCESS or ALLOW WRITE ACCESS option to allow other transactions either read-only or read-write access to the table while the indexes are being reorganized. While ALLOW READ ACCESS and ALLOW WRITE ACCESS allow access to the table, during the period in which the reorganized copies of the indexes are made available, no access to the table is allowed.

TABLE table-name
Specifies the table to reorganize. The table can be in a local or a remote database. The name or alias in the form: *schema.table-name* can be used. The *schema* is the user name under which the table was created. If you omit the schema name, the default schema is assumed.

For typed tables, the specified table name must be the name of the hierarchy's root table.

You cannot specify an index for the reorganization of a multidimensional clustering (MDC) table. In place reorganization of tables cannot be used for MDC tables.

INDEX index-name
Specifies the index to use when reorganizing the table. If you do not specify the fully qualified name in the form: *schema.index-name*, the default schema is assumed. The *schema* is the user name under which the index was created. The database manager uses the index to physically reorder the records in the table it is reorganizing.

For an in place table reorganization, if a clustering index is defined on the table and an index is specified, it must be clustering index. If the in place option is not specified, any index specified will be used. If you do not specify the name of an index, the records are reorganized without regard to order. If the table has a clustering index defined, however, and no index is specified, then the clustering index is used to cluster the table. You cannot specify an index if you are reorganizing an MDC table.

ALLOW NO ACCESS
Specifies that no other users can access the table while the table is being reorganized. When reorganizing a partitioned table, this is the default. Reorganization of a partitioned table occurs offline.

ALLOW READ ACCESS
Allow only read access to the table during reorganization. This is the default for a non-partitioned table.

INPLACE
Reorganizes the table while permitting user access.

In place table reorganization is allowed only on non-partitioned and non-MDC tables with type-2 indexes, but without extended indexes and with no indexes defined over XML columns in the table. In place table reorganization takes place asynchronously, and might not be effective immediately.

ALLOW READ ACCESS
Allow only read access to the table during reorganization.

ALLOW WRITE ACCESS
Allow write access to the table during reorganization. This is the default behavior.

NOTRUNCATE TABLE
Do not truncate the table after in place reorganization. During truncation, the table is S-locked.

START
Start the in place REORG processing. Because this is the default, this keyword is optional.

STOP Stop the in place REORG processing at its current point.

PAUSE
Suspend or pause in place REORG for the time being.

RESUME
Continue or resume a previously paused in place table reorganization. When an online reorganization is resumed and you want the same options as when the reorganization was paused, you must specify those options again while resuming.

USE tbspace-name
Specifies the name of a system temporary table space in which to store a temporary copy of the table being reorganized. If you do not provide a table space name, the database manager stores a working copy of the table in the table spaces that contain the table being reorganized.

For an 8KB, 16KB, or 32KB table object, if the page size of the system temporary table space that you specify does not match the page size of the table spaces in which the table data resides, the DB2 database product will try to find a temporary table space of the correct size of the LONG/LOB objects. Such a table space must exist for the reorganization to succeed.

When you have two temporary tablespaces of the same page size, and you specify one of them in the `USE` clause, they will be used in a round robin fashion if there is an index in the table being reorganized. Say you have two tablespaces, `tempsace1` and `tempspace2`, both of the same page size and you specify `tempspace1` in the `REORG` command with the `USE` option. When you perform `REORG` the first time, `tempspace1` is used. The second time, `tempspace2` is used. The third time, `tempspace1` is used and so on. To avoid this, you should drop one of the temporary tablespaces.

For partitioned tables, the table space is used as temporary storage for the reorganization of all the data partitions in the table. Reorganization of a partitioned table reorganizes a single data partition at a time. The amount of space required is equal to the largest data partition in the table, and not the entire table.

If you do not supply a table space name for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

INDEXSCAN
For a clustering REORG an index scan will be used to re-order table records. Reorganize table rows by accessing the table through an index. The default method is to scan the table and sort the result to reorganize the table, using temporary table spaces as necessary. Even though the index keys are in sort order, scanning and sorting is typically faster than fetching rows by first reading the row identifier from an index.

LONGLOBDATA
Long field and LOB data are to be reorganized.

This is not required even if the table contains long or LOB columns. The default is to avoid reorganizing these objects because it is time consuming and does not improve clustering.

USE longtbspace-name
This is an optional parameter, which can be used to specify the name of a temporary tablespace to be used for rebuilding long data. If no temporary tablespace is specified for either the table object or for the long objects, the objects will be constructed in the tablespace they currently reside. If a temporary tablespace is specified for the table but this parameter is not specified, then the tablespace used for base reorg data will be used, unless the page sizes differ. In this situation, the DB2 database system will attempt to choose a temporary container of the appropriate page size to create the long objects in.

If `USE-longtbspace` is specified, `USE-tbspace` must also be specified. If it is not, the `longtbspace` argument is ignored.

KEEPDICTIONARY
If the `COMPRESS` attribute for the table is `YES` and the table has a compression dictionary then no new dictionary is built. All the rows processed during reorganization are subject to compression using the existing dictionary. If the `COMPRESS` attribute for the table is `NO` and the table has a compression dictionary then reorg processing will remove the dictionary and all the rows in the newly reorganized table will be in non-compressed format. It is not possible to compress long, LOB, index, or XML objects.

Table 130. REORG KEEPDICTIONARY

Compress	Dictionary Exists	Result; outcome	
Y	Y	Preserve dictionary; rows compressed	
Y	N	Build dictionary; rows compressed	
N	Y	Preserve dictionary; all rows uncompressed	
N	N	No effect; all rows uncompressed	

For any reinitialization or truncation of a table (such as for a replace operation), if the compress attribute for the table is NO, the dictionary is discarded if one exists. Conversely, if a dictionary exists and the compress attribute for the table is YES then a truncation will save the dictionary and not discard it. The dictionary is logged in its entirety for recovery purposes and for future support with data capture changes (i.e. replication).

RESETDICTIONARY

If the `COMPRESS` attribute for the table is `YES` then a new row compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary. If the `COMPRESS` attribute for the table is `NO` and the table does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in non-compressed format. It is not possible to compress long, LOB, index, or XML objects.

Table 131. REORG RESETDICTIONARY

Compress	Dictionary Exists	Result; outcome	
Y	Y	Build new dictionary*; rows compressed	
Y	N	Build new dictionary; rows compressed	
N	Y	Remove dictionary; all rows uncompressed	
N	N	No effect; all rows uncompressed	

* - If a dictionary exists and the compression attribute is enabled but there is currently insufficient data in the table to build a new dictionary, the `RESETDICTIONARY` operation will keep the existing dictionary. Rows which are smaller in size than the internal minimum record length and rows which do not demonstrate a savings in record length when an attempt is made to compress them are considered ’insufficient’ in this case.

Usage notes:

Restrictions:

- The `REORG` utility does not support the use of nicknames.
- The `REORG TABLE` command is not supported for declared temporary tables.
- The `REORG TABLE` command cannot be used on views.
- Reorganization of a table is not compatible with range-clustered tables, because the range area of the table always remains clustered.
- `REORG TABLE` cannot be used on a partitioned table in a DMS table space while an online backup of ANY table space in which the table resides, including LOBs and indexes, is being performed.
- `REORG TABLE` cannot use an index that is based on an index extension.
- If a table is in reorg pending state, an inplace reorg is not allowed on the table.
- For partitioned tables:
  - REORG is supported at the table level. Reorganization of an individual data partition can be achieved by detaching the data partition, reorganizing the resulting non-partitioned table and then re-attaching the data partition.
  - The table must have an ACCESS_MODE in SYSCAT.TABLES of Full Access.
  - Reorganization skips data partitions that are in a restricted state due to an attach or detach operation
  - If an error occurs, the non-partitioned indexes of the table are marked bad, and are rebuilt on the next access to the table.
  - If a reorganization operation fails, some data partitions may be in a reorganized state and others may not. When the REORG TABLE command is reissued, all the data partitions will be reorganized regardless of the data partition's reorganization state.
  - When reorganizing indexes on partitioned tables, it is recommended that you perform a runstats operation after asynchronous index cleanup is complete in order to generate accurate index statistics in the presence of detached data partitions. To determine whether or not there are detached data partitions in the table, you can check the status field in SYSDATAPARTITIONS and look for the value ″I″ (index cleanup) or ″D″ (detached with dependant MQT).

Information about the current progress of table reorganization is written to the history file for database activity. The history file contains a record for each reorganization event. To view this file, execute the LIST HISTORY command for the database that contains the table you are reorganizing.

You can also use table snapshots to monitor the progress of table reorganization. Table reorganization monitoring data is recorded regardless of the Database Monitor Table Switch setting.

If an error occurs, an SQLCA dump is written to the history file. For an in-place table reorganization, the status is recorded as PAUSED.

When an indexed table has been modified many times, the data in the indexes might become fragmented. If the table is clustered with respect to an index, the table and index can get out of cluster order. Both of these factors can adversely affect the performance of scans using the index, and can impact the effectiveness of index page prefetching. REORG INDEX or REORG INDEXES can be used to reorganize one or all of the indexes on a table. Index reorganization will remove any fragmentation and restore physical clustering to the leaf pages. Use REORGCHK to help determine if an index needs reorganizing. Be sure to complete all database operations and release all locks before invoking index reorganization. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK.

Indexes might not be optimal following an in-place REORG TABLE operation, since only the data object and not the indexes are reorganized. It is recommended that you perform a REORG INDEXES after an in place REORG TABLE operation. Indexes are completely rebuilt during the last phase of a classic REORG TABLE, however, so reorganizing indexes is not necessary.

Tables that have been modified so many times that data is fragmented and access performance is noticeably slow are candidates for the REORG TABLE command. You should also invoke this utility after altering the inline length of a structured type column in order to benefit from the altered inline length. Use REORGCHK to determine whether a table needs reorganizing. Be sure to complete all database operations and release all locks before invoking REORG TABLE. This can be done by issuing a COMMIT after closing all cursors opened WITH HOLD, or by issuing a ROLLBACK. After reorganizing a table, use RUNSTATS to update the table statistics, and REBIND to rebind the packages that use this table. The reorganize utility will implicitly close all the cursors.

If the table contains mixed row format because the table value compression has been activated or deactivated, an offline table reorganization can convert all the existing rows into the target row format.

If the table is distributed across several database partitions, and the table or index reorganization fails on any of the affected database partitions, only the failing database partitions will have the table or index reorganization rolled back.

If the reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.

If the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is *not* specified, and if a clustering index exists, the data will be ordered according to the clustering index.

The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.

To complete a table space roll-forward recovery following a table reorganization, both regular and large table spaces must be enabled for roll-forward recovery.

If the table contains LOB columns that do not use the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS or DMS).

Related concepts:

- "Data row compression" in *Administration Guide: Implementation*
- "Table reorganization" in *Performance Guide*

Related reference:

- "db2Reorg - Reorganize an index or a table" on page 748

- “GET SNAPSHOT command” in *Command Reference*
- “REBIND ” on page 659
- “REORGCHK command” in *Command Reference*
- “REORG INDEXES/TABLE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function - Retrieve table reorganization snapshot information” in *Administrative SQL Routines and Views*
- “REORGCHK_TB_STATS procedure ÔÇô Retrieve table statistics for reorganization evaluation” in *Administrative SQL Routines and Views*
- “REORGCHK_IX_STATS procedure ÔÇô Retrieve index statistics for reorganization evaluation” in *Administrative SQL Routines and Views*

RESTART DATABASE

Restarts a database that has been abnormally terminated and left in an inconsistent state. At the successful completion of RESTART DATABASE, the application remains connected to the database if the user has CONNECT privilege.

Scope:

This command affects only the node on which it is executed.

Authorization:

None

Required connection:

This command establishes a database connection.

Command syntax:

```
►►──RESTART──┬─DATABASE─┬──database-alias──────────────────────────────►
             └─DB───────┘

►──┬──────────────────────────────────┬─────────────────────────────────►
   └─USER──username──┬──────────────┬─┘
                     └─USING──password─┘

►──┬────────────────────────────────────────────────────┬──┬──────────────┬──►◄
   │                             ┌─,──────────────┐      │  └─WRITE RESUME─┘
   │                             ▼                │      │
   └─DROP PENDING TABLESPACES──(───tablespace-name─┴──)──┘
```

Command parameters:

DATABASE database-alias
: Identifies the database to restart.

USER username
: Identifies the user name under which the database is to be restarted.

USING password
: The password used to authenticate *username*. If the password is omitted, the user is prompted to enter it.

DROP PENDING TABLESPACES tablespace-name
Specifies that the database restart operation is to be successfully completed even if table space container problems are encountered.

If a problem occurs with a container for a specified table space during the restart process, the corresponding table space will not be available (it will be in drop-pending state) after the restart operation. If a table space is in the drop-pending state, the only possible action is to drop the table space.

In the case of circular logging, a troubled table space will cause a restart failure. A list of troubled table space names can found in the administration notification log if a restart database operation fails because of container problems. If there is only one system temporary table space in the database, and it is in drop pending state, a new system temporary table space must be created immediately following a successful database restart operation.

WRITE RESUME
Allows you to force a database restart on databases that failed while I/O writes were suspended. Before performing crash recovery, this option will resume I/O writes by removing the SUSPEND_WRITE state from every table space in the database.

The WRITE RESUME option can also be used in the case where the connection used to suspend I/O writes is currently hung and all subsequent connection attempts are also hanging. When used in this circumstance, RESTART DATABASE will resume I/O writes to the database without performing crash recovery. RESTART DATABASE with the WRITE RESUME option will only perform crash recovery when you use it after a database crash. The WRITE RESUME parameter can only be applied to the primary database, not to mirrored databases.

Usage notes:

Execute this command if an attempt to connect to a database returns an error message, indicating that the database must be restarted. This action occurs only if the previous session with this database terminated abnormally (due to power failure, for example).

At the completion of RESTART DATABASE, a shared connection to the database is maintained if the user has CONNECT privilege, and an SQL warning is issued if any indoubt transactions exist. In this case, the database is still usable, but if the indoubt transactions are not resolved before the last connection to the database is dropped, another RESTART DATABASE must be issued before the database can be used again. Use the LIST INDOUBT TRANSACTIONS command to generate a list of indoubt transactions.

If the database is only restarted on a single node within an MPP system, a message might be returned on a subsequent database query indicating that the database needs to be restarted. This occurs because the database partition on a node on which the query depends must also be restarted. Restarting the database on all nodes solves the problem.

Related tasks:

- "Resolving indoubt transactions manually" in *Administration Guide: Planning*

Related reference:

- "LIST INDOUBT TRANSACTIONS command" in *Command Reference*
- "db2DatabaseRestart - Restart database" on page 679

RESTORE DATABASE

The `RESTORE DATABASE` command recreates a damaged or corrupted database that has been backed up using the DB2 backup utility. The restored database is in the same state that it was in when the backup copy was made. This utility can also overwrite a database with a different image or restore the backup copy to a new database.

For information on the restore operations supported by DB2 database systems between different operating systems and hardware platforms, see ″Backup and restore operations between different operating systems and hardware platforms″ in the *Related concepts* section.

The restore utility can also be used to restore backup images that were produced on DB2 UDB Verson 8. If a migration is required, it will be invoked automatically at the end of the restore operation.

If, at the time of the backup operation, the database was enabled for rollforward recovery, the database can be brought to its previous state by invoking the rollforward utility after successful completion of a restore operation.

This utility can also restore a table space level backup.

Incremental images and images only capturing differences from the time of the previous capture (called a "delta image") cannot be restored when there is a difference in operating systems or word size (32-bit or 64-bit).

Following a successful restore operation from one environment to a different environment, no incremental or delta backups are allowed until a non-incremental backup is taken. (This is not a limitation following a restore operation within the same environment.)

Even with a successful restore operation from one environment to a different environment, there are some considerations: packages must be rebound before use (using the BIND command, the REBIND command, or the db2rbind utility); SQL procedures must be dropped and re-created; and all external libraries must be rebuilt on the new platform. (These are not considerations when restoring to the same environment.)

Scope:

This command only affects the node on which it is executed.

Authorization:

To restore to an existing database, one of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:

- *sysadm*
- *sysctrl*

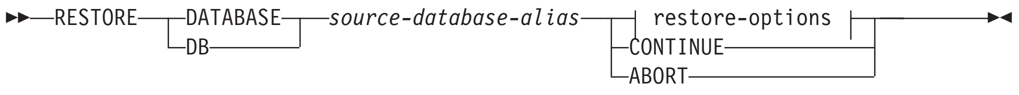
Required connection:

The required connection will vary based on the type of restore action:

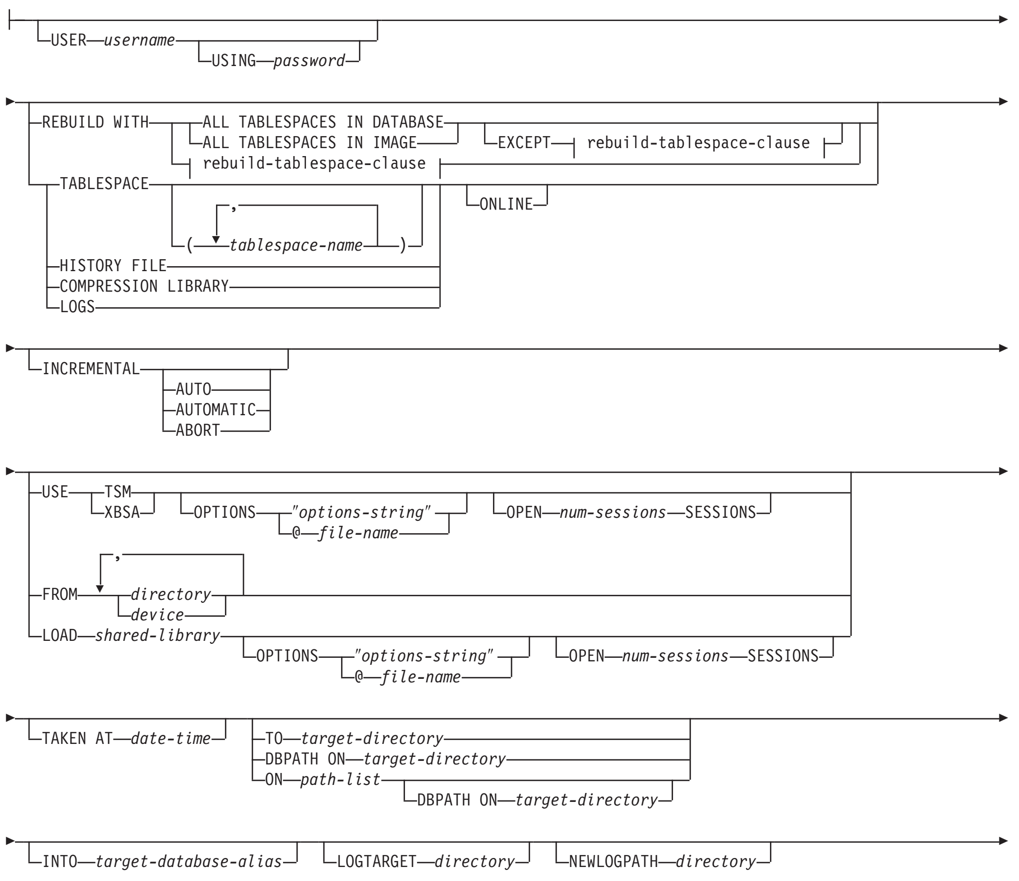
- You require a database connection, to restore to an existing database. This command automatically establishes an exclusive connection to the specified database.
- You require an instance and a database connection, to restore to a new database. The instance attachment is required to create the database.

  To restore to a new database at an instance different from the current instance, it is necessary to first attach to the instance where the new database will reside. The new instance can be local or remote. The current instance is defined by the value of the DB2INSTANCE environment variable.

Command syntax:



restore-options:



```
►─┬───────────────────────────┬──┬─────────────────────┬──┬───────────────────┬────────────►
  └─WITH──num-buffers──BUFFERS─┘  └─BUFFER──buffer-size─┘  └─DLREPORT──filename─┘

►─┬─────────────────────┬──┬──────────────────┬──┬─────────────────────────────────────┬────►
  └─REPLACE HISTORY FILE─┘  └─REPLACE EXISTING─┘  └─REDIRECT─┬──────────────────────────┬┘
                                                             └─GENERATE SCRIPT──script─┘

►─┬────────────────┬──┬───────────────┬──┬───────────────────┬──┬─────────────────────────┬─►
  └─PARALLELISM──n─┘  └─COMPRLIB──name─┘  └─COMPROPTS──string─┘  └─WITHOUT ROLLING FORWARD─┘

►─┬─────────────────┬──┬──────────────────┬───────────────────────────────────────────────►◄
  └─WITHOUT DATALINK─┘  └─WITHOUT PROMPTING─┘
```

rebuild-tablespace-clause:

```
                 ┌─,───────────────┐
                 ▼                 │
├──TABLESPACE──(───tablespace-name─┴──)──────────────────────────────────────┤
```

Command parameters:

DATABASE *source-database-alias*
: Alias of the source database from which the backup was taken.

CONTINUE
: Specifies that the containers have been redefined, and that the final step in a redirected restore operation should be performed.

ABORT
: This parameter:
  - Stops a redirected restore operation. This is useful when an error has occurred that requires one or more steps to be repeated. After RESTORE DATABASE with the ABORT option has been issued, each step of a redirected restore operation must be repeated, including RESTORE DATABASE with the REDIRECT option.
  - Terminates an incremental restore operation before completion.

USER *username*
: Identifies the user name under which the database is to be restored.

USING *password*
: The password used to authenticate the user name. If the password is omitted, the user is prompted to enter it.

REBUILD WITH ALL TABLESPACES IN DATABASE
: Restores the database with all the table spaces known to the database at the time of the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN DATABASE EXCEPT *rebuild-tablespace-clause*
: Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE
: Restores the database with only the table spaces in the image being restored. This restore overwrites a database if it already exists.

REBUILD WITH ALL TABLESPACES IN IMAGE EXCEPT *rebuild-tablespace-clause*
: Restores the database with only the table spaces in the image being restored except for those specified in the list. This restore overwrites a database if it already exists.

REBUILD WITH *rebuild-tablespace-clause*
: Restores the database with only the list of table spaces specified. This restore overwrites a database if it already exists.

TABLESPACE *tablespace-name*
: A list of names used to specify the table spaces that are to be restored.

ONLINE
: This keyword, applicable only when performing a table space-level restore operation, is specified to allow a backup image to be restored online. This means that other agents can connect to the database while the backup image is being restored, and that the data in other table spaces will be available while the specified table spaces are being restored.

HISTORY FILE
: This keyword is specified to restore only the history file from the backup image.

COMPRESSION LIBRARY
: This keyword is specified to restore only the compression library from the backup image. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.

LOGS
: This keyword is specified to restore only the set of log files contained in the backup image. If the backup image does not contain any log files, the restore operation will fail. If this option is specified, the LOGTARGET option must also be specified.

INCREMENTAL
: Without additional parameters, INCREMENTAL specifies a manual cumulative restore operation. During manual restore the user must issue each restore command manually for each image involved in the restore. Do so according to the following order: last, first, second, third and so on up to and including the last image.

INCREMENTAL AUTOMATIC/AUTO
: Specifies an automatic cumulative restore operation.

INCREMENTAL ABORT
: Specifies abortion of an in-progress manual cumulative restore operation.

USE TSM
: Specifies that the database is to be restored from TSM-managed output.

OPTIONS

"options-string"
: Specifies options to be used for the restore operation.The string will be passed to the vendor support library, for example TSM, exactly as it was entered, without the quotes. Specifying this option overrides the value specified by the VENDOROPT database configuration parameter.

@file-name
: Specifies that the options to be used for the restore operation are contained in a file located on the DB2 server. The string will be passed to the vendor support library, for example TSM. The file must be a fully qualified file name.

OPEN *num-sessions* **SESSIONS**
Specifies the number of I/O sessions that are to be used with TSM or the vendor product.

USE XBSA
Specifies that the XBSA interface is to be used. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.

FROM *directory/device*
The fully qualified path name of the directory or device on which the backup image resides. If USE TSM, FROM, and LOAD are omitted, the default value is the current working directory of the client machine. This target directory or device must exist on the target server/instance.

On Windows operating systems, the specified directory must not be a DB2-generated directory. For example, given the following commands:

```
db2 backup database sample to c:\backup
db2 restore database sample from c:\backup
```

Using these commands, the DB2 database system generates subdirectories under the `c:\backup` directory to allow more than one backup to be placed in the specified top level directory. The DB2 generated subdirectories should be ignored. To specify precisely which backup image to restore, use the TAKEN AT parameter. There can be several backup images stored on the same path.

If several items are specified, and the last item is a tape device, the user is prompted for another tape. Valid response options are:

c Continue. Continue using the device that generated the warning message (for example, continue when a new tape has been mounted).

d Device terminate. Stop using *only* the device that generated the warning message (for example, terminate when there are no more tapes).

t Terminate. Abort the restore operation after the user has failed to perform some action requested by the utility.

LOAD *shared-library*
The name of the shared library (DLL on Windows operating systems) containing the vendor backup and restore I/O functions to be used. The name can contain a full path. If the full path is not given, the value defaults to the path on which the user exit program resides.

TAKEN AT *date-time*
The time stamp of the database backup image. The time stamp is displayed after successful completion of a backup operation, and is part of the path name for the backup image. It is specified in the form *yyyymmddhhmmss*. A partial time stamp can also be specified. For example, if two different backup images with time stamps 20021001010101 and 20021002010101 exist, specifying `20021002` causes the image with time stamp `20021002010101` to be used. If a value for this parameter is not specified, there must be only one backup image on the source media.

TO *target-directory*
This parameter states the target database directory. This parameter is

ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local. If the backup image contains a database that is enabled for automatic storage then only the database directory changes, the storage paths associated with the database do not change.

DBPATH ON *target-directory*

This parameter states the target database directory. This parameter is ignored if the utility is restoring to an existing database. The drive and directory that you specify must be local. If the backup image contains a database that is enabled for automatic storage and the ON parameter is not specified then this parameter is synonymous with the TO parameter and only the database directory changes, the storage paths associated with the database do not change.

ON *path-list*

This parameter redefines the storage paths associated with an automatic storage database. Using this parameter with a database that is not enabled for automatic storage results in an error (SQL20321N). The existing storage paths as defined within the backup image are no longer used and automatic storage table spaces are automatically redirected to the new paths. If this parameter is not specified for an automatic storage database then the storage paths remain as they are defined within the backup image.

One or more paths can be specified, each separated by a comma. Each path must have an absolute path name and it must exist locally. If the database does not already exist on disk and the DBPATH ON parameter is not specified then the first path is used as the target database directory.

For a multi-partition database the ON path-list option can only be specified on the catalog partition. The catalog partition must be restored before any other partitions are restored when the ON option is used. The restore of the catalog-partition with new storage paths will place all non-catalog nodes in a RESTORE_PENDING state. The non-catalog nodes can then be restored in parallel without specifying the ON clause in the restore command.

In general, the same storage paths must be used for each partition in a multi-partition database and they must all exist prior to executing the `RESTORE DATABASE` command. One exception to this is where database partition expressions are used within the storage path. Doing this allows the database partition number to be reflected in the storage path such that the resulting path name is different on each partition.

You use the argument `" $N"` (`[blank]$N`) to indicate a database partition expression. A database partition expression can be used anywhere in the storage path, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the storage path after the database partition expression is evaluated. If there is no space character in the storage path after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms:

Table 132. . Operators are evaluated from left to right. `%` represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N	" $N"	10

Table 132. (continued). Operators are evaluated from left to right. % represents the modulus operator. The database partition number in the examples is assumed to be 10.

Syntax	Example	Value
[blank]$N+[number]	" $N+100"	110
[blank]$N%[number]	" $N%5"	0
[blank]$N+[number]%[number]	" $N+1%5"	1
[blank]$N%[number]+[number]	" $N%4+2"	4
[a] % is modulus.		

INTO *target-database-alias*
The target database alias. If the target database does not exist, it is created.

When you restore a database backup to an existing database, the restored database inherits the alias and database name of the existing database. When you restore a database backup to a nonexistent database, the new database is created with the alias and database name that you specify. This new database name must be unique on the system where you restore it.

LOGTARGET *directory*
The absolute path name of an existing directory on the database server, to be used as the target directory for extracting log files from a backup image. If this option is specified, any log files contained within the backup image will be extracted into the target directory. If this option is not specified, log files contained within a backup image will not be extracted. To extract only the log files from the backup image, specify the LOGS option.

NEWLOGPATH *directory*
The absolute pathname of a directory that will be used for active log files after the restore operation. This parameter has the same function as the *newlogpath* database configuration parameter, except that its effect is limited to the restore operation in which it is specified. The parameter can be used when the log path in the backup image is not suitable for use after the restore operation; for example, when the path is no longer valid, or is being used by a different database.

WITH *num-buffers* **BUFFERS**
The number of buffers to be used. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. A larger number of buffers can be used to improve performance when multiple sources are being read from, or if the value of PARALLELISM has been increased.

BUFFER *buffer-size*
The size, in pages, of the buffer used for the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value. The minimum value for this parameter is 8 pages.

The restore buffer size must be a positive integer multiple of the backup buffer size specified during the backup operation. If an incorrect buffer size is specified, the buffers are allocated to be of the smallest acceptable size.

DLREPORT *filename*
The file name, if specified, must be specified as an absolute path. Reports the files that become unlinked, as a result of a fast reconcile, during a restore operation. This option is only to be used if the table being restored has a DATALINK column type and linked files.

REPLACE HISTORY FILE
Specifies that the restore operation should replace the history file on disk with the history file from the backup image.

REPLACE EXISTING
If a database with the same alias as the target database alias already exists, this parameter specifies that the restore utility is to replace the existing database with the restored database. This is useful for scripts that invoke the restore utility, because the command line processor will not prompt the user to verify deletion of an existing database. If the WITHOUT PROMPTING parameter is specified, it is not necessary to specify REPLACE EXISTING, but in this case, the operation will fail if events occur that normally require user intervention.

REDIRECT
Specifies a redirected restore operation. To complete a redirected restore operation, this command should be followed by one or more SET TABLESPACE CONTAINERS commands, and then by a RESTORE DATABASE command with the CONTINUE option. All commands associated with a single redirected restore operation must be invoked from the same window or CLP session. A redirected restore operation cannot be performed against a table space that has automatic storage enabled.

GENERATE SCRIPT *script*
Creates a redirect restore script with the specified file name. The script name can be relative or absolute and the script will be generated on the client side. If the file cannot be created on the client side, an error message (SQL9304N) will be returned. If the file already exists, it will be overwritten. Please see the examples below for further usage information.

WITHOUT ROLLING FORWARD
Specifies that the database is not to be put in rollforward pending state after it has been successfully restored.

If, following a successful restore operation, the database is in rollforward pending state, the ROLLFORWARD command must be invoked before the database can be used again.

If this option is specified when restoring from an online backup image, error SQL2537N will be returned.

If backup image is of a recoverable database then WITHOUT ROLLING FORWARD cannot be specified with REBUILD option.

WITHOUT DATALINK
Specifies that any tables with DATALINK columns are to be put in DataLink_Reconcile_Pending (DRP) state, and that no reconciliation of linked files is to be performed.

PARALLELISM *n*
Specifies the number of buffer manipulators that are to be created during the restore operation. The DB2 database system will automatically choose an optimal value for this parameter unless you explicitly enter a value.

COMPRLIB *name*
Indicates the name of the library to be used to perform the decompression. The name must be a fully qualified path referring to a file on the server. If this parameter is not specified, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library cannot be loaded, the restore operation will fail.

COMPROPTS *string*

Describes a block of binary data that is passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte reversal or code page conversion are handled by the decompression library. If the first character of the data block is “@”, the remainder of the data is interpreted by the DB2 database system as the name of a file residing on the server. The DB2 database system will then replace the contents of *string* with the contents of this file and pass the new value to the initialization routine instead. The maximum length for the string is 1 024 bytes.

WITHOUT PROMPTING

Specifies that the restore operation is to run unattended. Actions that normally require user intervention will return an error message. When using a removable media device, such as tape or diskette, the user is prompted when the device ends, even if this option is specified.

Examples:

1. In the following example, the database WSDB is defined on all 4 database partitions, numbered 0 through 3. The path /dev3/backup is accessible from all database partitions. The following offline backup images are available from /dev3/backup:

   ```
   wsdb.0.db2inst1.NODE0000.CATN0000.20020331234149.001
   wsdb.0.db2inst1.NODE0001.CATN0000.20020331234427.001
   wsdb.0.db2inst1.NODE0002.CATN0000.20020331234828.001
   wsdb.0.db2inst1.NODE0003.CATN0000.20020331235235.001
   ```

   To restore the catalog partition first, then all other database partitions of the WSDB database from the /dev3/backup directory, issue the following commands from one of the database partitions:

   ```
   db2_all '<<+0< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331234149
     INTO wsdb REPLACE EXISTING'
   db2_all '<<+1< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331234427
     INTO wsdb REPLACE EXISTING'
   db2_all '<<+2< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331234828
     INTO wsdb REPLACE EXISTING'
   db2_all '<<+3< db2 RESTORE DATABASE wsdb FROM /dev3/backup
   TAKEN AT 20020331235235
     INTO wsdb REPLACE EXISTING'
   ```

   The `db2_all` utility issues the restore command to each specified database partition. When performing a restore using `db2_all`, you should always specify `REPLACE EXISTING` and/or `WITHOUT PROMPTING`. Otherwise, if there is prompting, the operation will look like it is hanging. This is because `db2_all` does not support user prompting.

2. Following is a typical redirected restore scenario for a database whose alias is MYDB:

   a. Issue a RESTORE DATABASE command with the REDIRECT option.

      ```
      restore db mydb replace existing redirect
      ```

      After successful completion of step 1, and before completing step 3, the restore operation can be aborted by issuing:

      ```
      restore db mydb abort
      ```

b. Issue a SET TABLESPACE CONTAINERS command for each table space whose containers must be redefined. For example:

```
set tablespace containers for 5 using
   (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
```

To verify that the containers of the restored database are the ones specified in this step, issue the LIST TABLESPACE CONTAINERS command.

c. After successful completion of steps 1 and 2, issue:

```
restore db mydb continue
```

This is the final step of the redirected restore operation.

d. If step 3 fails, or if the restore operation has been aborted, the redirected restore can be restarted, beginning at step 1.

3. Following is a sample weekly incremental backup strategy for a recoverable database. It includes a weekly full database backup operation, a daily non-cumulative (delta) backup operation, and a mid-week cumulative (incremental) backup operation:

```
(Sun) backup db mydb use tsm
(Mon) backup db mydb online incremental delta use tsm
(Tue) backup db mydb online incremental delta use tsm
(Wed) backup db mydb online incremental use tsm
(Thu) backup db mydb online incremental delta use tsm
(Fri) backup db mydb online incremental delta use tsm
(Sat) backup db mydb online incremental use tsm
```

For an automatic database restore of the images created on Friday morning, issue:

```
restore db mydb incremental automatic taken at (Fri)
```

For a manual database restore of the images created on Friday morning, issue:

```
restore db mydb incremental taken at (Fri)
restore db mydb incremental taken at (Sun)
restore db mydb incremental taken at (Wed)
restore db mydb incremental taken at (Thu)
restore db mydb incremental taken at (Fri)
```

4. To produce a backup image, which includes logs, for transportation to a remote site:

```
backup db sample online to /dev3/backup include logs
```

To restore that backup image, supply a LOGTARGET path and specify this path during ROLLFORWARD:

```
restore db sample from /dev3/backup logtarget /dev3/logs
rollforward db sample to end of logs and stop overflow log path /dev3/logs
```

5. To retrieve only the log files from a backup image that includes logs:

```
restore db sample logs from /dev3/backup logtarget /dev3/logs
```

6. The USE TSM OPTIONS keywords can be used to specify the TSM information to use for the restore operation. On Windows platforms, omit the -fromowner option.

   - Specifying a delimited string:

```
restore db sample use TSM options '"-fromnode=bar -fromowner=dmcinnis"'
```

   - Specifying a fully qualified file:

```
restore db sample use TSM options @/u/dmcinnis/myoptions.txt
```

The file myoptions.txt contains the following information: -fromnode=bar -fromowner=dmcinnis

7. The following is a simple restore of a multi-partition automatic storage enabled database with new storage paths. The database was originally created with one storage path, `/myPath0`:
   - On the catalog partition issue: `restore db mydb on /myPath1,/myPath2`
   - On all non-catalog partitions issue: `restore db mydb`
8. A script output of the following command on a non-auto storage database:

```
restore db sample from /home/jseifert/backups taken at 20050301100417 redirect
generate script SAMPLE_NODE0000.clp
```

would look like this:

```
-- ****************************************************************************
-- ** automatically created redirect restore script
-- ****************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON SAMPLE_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- ****************************************************************************
-- ** initialize redirected restore
-- ****************************************************************************
RESTORE DATABASE SAMPLE
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050301100417
-- DBPATH ON '<target-directory>'
INTO SAMPLE
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00001/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- ****************************************************************************
-- ** tablespace definition
-- ****************************************************************************
-- ****************************************************************************
-- ** Tablespace name                            = SYSCATSPACE
-- **   Tablespace ID                            = 0
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 5572
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 0
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0000.0'
);
-- ****************************************************************************
-- ** Tablespace name                            = TEMPSPACE1
-- **   Tablespace ID                            = 1
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = System Temporary data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 0
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 1
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0001.0'
);
-- ****************************************************************************
-- ** Tablespace name                            = USERSPACE1
-- **   Tablespace ID                            = 2
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Total number of pages                    = 1
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 2
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  PATH   'SQLT0002.0'
);
-- ****************************************************************************
-- ** Tablespace name                            = DMS
-- **   Tablespace ID                            = 3
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Auto-resize enabled                      = No
-- **   Total number of pages                    = 2000
-- **   Number of usable pages                   = 1960
-- **   High water mark (pages)                  = 96
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   /tmp/dms1                                                   1000
, FILE   /tmp/dms2                                                   1000
);
-- ****************************************************************************
-- ** Tablespace name                            = RAW
-- **   Tablespace ID                            = 4
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = No
-- **   Auto-resize enabled                      = No
-- **   Total number of pages                    = 2000
-- **   Number of usable pages                   = 1960
-- **   High water mark (pages)                  = 96
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                             1000
, DEVICE '/dev/hdb2'                                             1000
);
-- ****************************************************************************
-- ** start redirect restore
-- ****************************************************************************
RESTORE DATABASE SAMPLE CONTINUE;
```

```
-- ****************************************************************************
-- ** end of file
-- ****************************************************************************
```

9. A script output of the following command on an automatic storage database:

```
restore db test from /home/jseifert/backups taken at 20050304090733 redirect
generate script TEST_NODE0000.clp
```

would look like this:

```
-- ****************************************************************************
-- ** automatically created redirect restore script
-- ****************************************************************************
UPDATE COMMAND OPTIONS USING S ON Z ON TEST_NODE0000.out V ON;
SET CLIENT ATTACH_DBPARTITIONNUM  0;
SET CLIENT CONNECT_DBPARTITIONNUM 0;
-- ****************************************************************************
-- ** initialize redirected restore
-- ****************************************************************************
RESTORE DATABASE TEST
-- USER  '<username>'
-- USING '<password>'
FROM '/home/jseifert/backups'
TAKEN AT 20050304090733
ON '/home/jseifert'
-- DBPATH ON <target-directory>
INTO TEST
-- NEWLOGPATH '/home/jseifert/jseifert/NODE0000/SQL00002/SQLOGDIR/'
-- WITH <num-buff> BUFFERS
-- BUFFER <buffer-size>
-- REPLACE HISTORY FILE
-- REPLACE EXISTING
REDIRECT
-- PARALLELISM <n>
-- WITHOUT ROLLING FORWARD
-- WITHOUT PROMPTING
;
-- ****************************************************************************
-- ** tablespace definition
-- ****************************************************************************
-- ****************************************************************************
-- ** Tablespace name                            = SYSCATSPACE
-- **   Tablespace ID                            = 0
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                  = Any data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 4
-- **   Using automatic storage                  = Yes
-- **   Auto-resize enabled                      = Yes
-- **   Total number of pages                    = 6144
-- **   Number of usable pages                   = 6140
-- **   High water mark (pages)                  = 5968
-- ****************************************************************************
-- ****************************************************************************
-- ** Tablespace name                            = TEMPSPACE1
-- **   Tablespace ID                            = 1
-- **   Tablespace Type                          = System managed space
-- **   Tablespace Content Type                  = System Temporary data
-- **   Tablespace Page size (bytes)             = 4096
-- **   Tablespace Extent size (pages)           = 32
-- **   Using automatic storage                  = Yes
-- **   Total number of pages                    = 0
-- ****************************************************************************
-- ****************************************************************************
-- ** Tablespace name                            = USERSPACE1
-- **   Tablespace ID                            = 2
-- **   Tablespace Type                          = Database managed space
-- **   Tablespace Content Type                   = Any data
-- **   Tablespace Page size (bytes)              = 4096
-- **   Tablespace Extent size (pages)            = 32
-- **   Using automatic storage                   = Yes
-- **   Auto-resize enabled                       = Yes
-- **   Total number of pages                     = 256
-- **   Number of usable pages                    = 224
-- **   High water mark (pages)                   = 96
-- ****************************************************************************
-- ****************************************************************************
-- ** Tablespace name                             = DMS
-- **   Tablespace ID                             = 3
-- **   Tablespace Type                           = Database managed space
-- **   Tablespace Content Type                   = Any data
-- **   Tablespace Page size (bytes)              = 4096
-- **   Tablespace Extent size (pages)            = 32
-- **   Using automatic storage                   = No
-- **   Auto-resize enabled                       = No
-- **   Total number of pages                     = 2000
-- **   Number of usable pages                    = 1960
-- **   High water mark (pages)                   = 96
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 3
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  FILE   '/tmp/dms1'                                         1000
, FILE   '/tmp/dms2'                                         1000
);
-- ****************************************************************************
-- ** Tablespace name                             = RAW
-- **   Tablespace ID                             = 4
-- **   Tablespace Type                           = Database managed space
-- **   Tablespace Content Type                   = Any data
-- **   Tablespace Page size (bytes)              = 4096
-- **   Tablespace Extent size (pages)            = 32
-- **   Using automatic storage                   = No
-- **   Auto-resize enabled                       = No
-- **   Total number of pages                     = 2000
-- **   Number of usable pages                    = 1960
-- **   High water mark (pages)                   = 96
-- ****************************************************************************
SET TABLESPACE CONTAINERS FOR 4
-- IGNORE ROLLFORWARD CONTAINER OPERATIONS
USING (
  DEVICE '/dev/hdb1'                                         1000
, DEVICE '/dev/hdb2'                                         1000
);
-- ****************************************************************************
-- ** start redirect restore
-- ****************************************************************************
RESTORE DATABASE TEST CONTINUE;
-- ****************************************************************************
-- ** end of file
-- ****************************************************************************
```

Usage notes:

- A RESTORE DATABASE command of the form `db2 restore db <name>` will perform a full database restore with a database image and will perform a table space restore operation of the table spaces found in a table space image. A RESTORE DATABASE command of the form `db2 restore db <name> tablespace` performs a table space restore of the table spaces found in the image. In addition, if a list of table spaces is provided with such a command, the explicitly listed table spaces are restored.

- Following the restore operation of an online backup, you must perform a roll-forward recovery.
- If a backup image is compressed, the DB2 database system detects this and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. Otherwise, a check is made to see if a library is stored in the backup image and if it exists it is used. Finally, if there is not library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is to be restored from a backup image (either explicitly by specifying the `COMPRESSION LIBRARY` option or implicitly by performing a normal restore of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was taken on. If the platform the backup was taken on is not the same as the platform that the restore is being done on, the restore operation will fail, even if DB2 normally supports cross-platform restores involving the two systems.
- To restore log files from the backup image that contains them, the LOGTARGET option must be specified, providing the fully qualified and valid path that exists on the DB2 server. If those conditions are satisfied, the restore utility will write the log files from the image to the target path. If a LOGTARGET is specified during a restore of a backup image that does not include logs, the restore oepration will return an error before attempting to restore any table space data. A restore operation will also fail with an error if an invalid, or read-only, LOGTARGET path is specified.
- If any log files exist in the LOGTARGET path at the time the RESTORE DATABASE command is issued, a warning prompt will be returned to the user. This warning will not be returned if WITHOUT PROMPTING is specified.
- During a restore operation where a LOGTARGET is specified, if any log file cannot be extracted, the restore operation will fail and return an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation will fail and an error will be returned. The restore database utility will not overwrite existing log files in the LOGTARGET directory.
- You can also restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path will result in an error. If any problem occurs while restoring log files in this mode of operation, the restore operation will terminate immediately and an error will be returned.
- During an automatic incremental restore operation, only the log files included in the target image of the restore operation will be retrived from the backup image. Any log files included in intermediate images referenced during the incremental restore process will not be extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should only be specified with the final restore command to be issued.
- A backup targeted to be restored to another operating system or another DB2 database version must be an offline backup, and cannot be a delta or an incremental backup image. The same is true for backups to be restored to a later DB2 database version.

Related concepts:

- "Backup and restore operations between different operating systems and hardware platforms" in *Data Recovery and High Availability Guide and Reference*

- “Developing a backup and recovery strategy” in *Data Recovery and High Availability Guide and Reference*

Related tasks:

- “Using restore” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “CREATE DATABASE ” on page 461
- “db2move - Database movement tool command” in *Command Reference*

ROLLFORWARD DATABASE

Recovers a database by applying transactions recorded in the database log files. Invoked after a database or a table space backup image has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, the *logarchmeth1* or *logarchmeth2* database configuration parameters must be set to a value other than `OFF`) before the database can be recovered with rollforward recovery.

Scope:

In a partitioned database environment, this command can only be invoked from the catalog partition. A database or table space rollforward operation to a specified point in time affects all database partitions that are listed in the `db2nodes.cfg` file. A database or table space rollforward operation to the end of logs affects the database partitions that are specified. If no database partitions are specified, it affects all database partitions that are listed in the `db2nodes.cfg` file; if rollforward recovery is not needed on a particular partition, that partition is ignored.

For partitioned tables, you are also required to roll forward related table spaces to the same point in time. This applies to table spaces containing data partitions of a table. If a single table space contains a portion of a partitioned table, rolling forward to the end of the logs is still allowed.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None. This command establishes a database connection.

Command syntax:

```
►►──ROLLFORWARD──┬─DATABASE─┬──database-alias──┬──────────────────────────────────┬──────────►
                 └─DB───────┘                  └─USER──username──┬──────────────────┬┘
                                                                 └─USING──password──┘
```

```
►─┬──────────────────────────────────────────────────────────────────────────────────┬──►
  ├─TO──┬─isotime──┬─────────────────────┬─┬──────────────────────┬─┬─┬──────────────┬─┤
  │     │          ├─USING UTC TIME──────┤ │                      │ │ ├─AND COMPLETE─┤ │
  │     │          └─USING LOCAL TIME────┘ └─ON ALL DBPARTITIONNUMS┘ │ └─AND STOP─────┘ │
  │     └─END OF LOGS──┬──────────────────────────────┬───────────────┘                  │
  │                    └─┤ On Database Partition clause ├─┘                              │
  └─┬─COMPLETE──────────────────────────┬─┬──────────────────────────────┬──────────────┘
    ├─STOP──────────────────────────────┤ └─┤ On Database Partition clause ├─┘
    ├─CANCEL────────────────────────────┤
    └─QUERY STATUS──┬─────────────────┬─┘
                    ├─USING UTC TIME──┤
                    └─USING LOCAL TIME┘

►─┬──────────────────────────────────────────────────┬──────────────────────────────────►
  └─TABLESPACE──┬─ONLINE─────────────────────────────┤
                │    ┌─,───────────────┐             │
                └─(──▼─tablespace-name─┴──)──┬────────┤
                                             └─ONLINE─┘

►─┬─────────────────────────────────────────────────────────────┬──┬────────────┬────────►
  └─OVERFLOW LOG PATH──(──log-directory──┬──────────────────────┬──)─┘  └─NORETRIEVE─┘
                                         └─,─┤ Log Overflow clause ├─┘

►─┬─────────────────────────────────────────────────────────┬──────────────────────────►◄
  └─RECOVER DROPPED TABLE──drop-table-id──TO──export-directory─┘
```

On Database Partition clause:

```
├──ON──┬─┤ Database Partition List clause ├──────────────────────────────┬───────────┤
       └─ALL DBPARTITIONNUMS──┬──────────────────────────────────────────┤
                              └─EXCEPT─┤ Database Partition List clause ├─┘
```

Database Partition List clause:

```
├──┬─DBPARTITIONNUM──┬──(─────────────────────────────────────────────────────────────►
   └─DBPARTITIONNUMS─┘

   ┌─,────────────────────────────────────────────────┐
►──▼─db-partition-number1──┬──────────────────────────┬─┴──)──────────────────────────┤
                           └─TO──db-partition-number2─┘
```

Log Overflow clause:

```
    ┌─,────────────────────────────────────────────────────────┐
├───▼─log-directory──ON DBPARTITIONNUM──db-partition-number1───┴──────────────────────┤
```

Command parameters:

DATABASE database-alias
The alias of the database that is to be rollforward recovered.

USER username
The user name under which the database is to be rollforward recovered.

USING password
The password used to authenticate the user name. If the password is omitted, you will be prompted to enter it.

TO

isotime

The point in time to which all committed transactions are to be rolled forward (including the transaction committed precisely at that time, as well as all transactions committed previously).

This value is specified as a time stamp, a 7-part character string that identifies a combined date and time. The format is *yyyy-mm-dd-hh.mm.ss* (year, month, day, hour, minutes, seconds), expressed in Coordinated Universal Time (UTC, formerly known as GMT). UTC helps to avoid having the same time stamp associated with different logs (because of a change in time associated with daylight savings time, for example). The time stamp in a backup image is based on the local time at which the backup operation started. The CURRENT TIMEZONE special register specifies the difference between UTC and local time at the application server. The difference is represented by a time duration (a decimal number in which the first two digits represent the number of hours, the next two digits represent the number of minutes, and the last two digits represent the number of seconds). Subtracting CURRENT TIMEZONE from a local time converts that local time to UTC.

USING LOCAL TIME

Allows you to rollforward to a point in time that is the server's local time rather than UTC time.

Notes:

1. If you specify a local time for rollforward, all messages returned to you will also be in local time. All times are converted on the server, and in partitioned database environments, on the catalog database partition.
2. The timestamp string is converted to UTC on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client.
3. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.
4. Subsequent ROLLFORWARD commands that cannot specify the USING LOCAL TIME clause will have all messages returned to you in local time if this option is specified.

END OF LOGS

Specifies that all committed transactions from all online archive log files listed in the database configuration parameter *logpath* are to be applied.

ALL DBPARTITIONNUMS

Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file. This is the default if a database partition clause is not specified.

EXCEPT

Specifies that transactions are to be rolled forward on all database partitions specified in the `db2nodes.cfg` file, except those specified in the database partition list.

ON DBPARTITIONNUM / ON DBPARTITIONNUMS
Roll the database forward on a set of database partitions.

db-partition-number1
Specifies a database partition number in the database partition list.

db-partition-number2
Specifies the second database partition number, so that all database partitions from *db-partition-number1* up to and including *db-partition-number2* are included in the database partition list.

COMPLETE / STOP
Stops the rolling forward of log records, and completes the rollforward recovery process by rolling back any incomplete transactions and turning off the rollforward pending state of the database. This allows access to the database or table spaces that are being rolled forward. These keywords are equivalent; specify one or the other, but not both. The keyword AND permits specification of multiple operations at once; for example, `db2 rollforward db sample to end of logs and complete`. When rolling table spaces forward to a point in time, the table spaces are placed in backup pending state.

CANCEL
Cancels the rollforward recovery operation. This puts the database or one or more table spaces on all database partitions on which forward recovery has been started in restore pending state:

- If a *database* rollforward operation is not in progress (that is, the database is in rollforward pending state), this option puts the database in restore pending state.
- If a *table space* rollforward operation is not in progress (that is, the table spaces are in rollforward pending state), a table space list must be specified. All table spaces in the list are put in restore pending state.
- If a table space rollforward operation *is* in progress (that is, at least one table space is in rollforward in progress state), all table spaces that are in rollforward in progress state are put in restore pending state. If a table space list is specified, it must include all table spaces that are in rollforward in progress state. All table spaces on the list are put in restore pending state.
- If rolling forward to a point in time, any table space name that is passed in is ignored, and all table spaces that are in rollforward in progress state are put in restore pending state.
- If rolling forward to the end of the logs with a table space list, only the table spaces listed are put in restore pending state.

This option cannot be used to cancel a rollforward operation *that is actually running*. It can only be used to cancel a rollforward operation that is in progress but not actually running at the time. A rollforward operation can be in progress but not running if:

- It terminated abnormally.
- The STOP option was not specified.
- An error caused it to fail. Some errors, such as rolling forward through a non-recoverable load operation, can put a table space into restore pending state.

Use this option with caution, and only if the rollforward operation that is in progress cannot be completed because some of the table spaces have been put in rollforward pending state or in restore pending state. When in doubt, use the LIST TABLESPACES command to identify the table spaces that are in rollforward in progress state, or in rollforward pending state.

QUERY STATUS
Lists the log files that the database manager has rolled forward, the next archive file required, and the time stamp (in UTC) of the last committed transaction since rollforward processing began. In a partitioned database environment, this status information is returned for each database partition. The information returned contains the following fields:

Database partition number

Rollforward status
Status can be: database or table space rollforward pending, database or table space rollforward in progress, database or table space rollforward processing STOP, or not pending.

Next log file to be read
A string containing the name of the next required log file. In a partitioned database environment, use this information if the rollforward utility fails with a return code indicating that a log file is missing or that a log information mismatch has occurred.

Log files processed
A string containing the names of processed log files that are no longer needed for recovery, and that can be removed from the directory. If, for example, the oldest uncommitted transaction starts in log file *x*, the range of obsolete log files will not include *x*; the range ends at *x* - 1.

Last committed transaction
A string containing a time stamp in ISO format (*yyyy-mm-dd-hh.mm.ss*) suffixed by either “UTC” or “Local” (see USING LOCAL TIME). This time stamp marks the last transaction committed after the completion of rollforward recovery. The time stamp applies to the database. For table space rollforward recovery, it is the time stamp of the last transaction committed to the database.

QUERY STATUS is the default value if the TO, STOP, COMPLETE, or CANCEL clauses are omitted. If TO, STOP, or COMPLETE was specified, status information is displayed if the command has completed successfully. If individual table spaces are specified, they are ignored; the status request does not apply only to specified table spaces.

TABLESPACE
This keyword is specified for table space-level rollforward recovery.

tablespace-name
Mandatory for table space-level rollforward recovery to a point in time. Allows a subset of table spaces to be specified for rollforward recovery to the end of the logs. In a partitioned database environment, each table space in the list does not have to exist at each database partition that is rolling forward. If it *does* exist, it must be in the correct state.

For partitioned tables, point in time roll-forward of a table space containing any piece of a partitioned table must also roll-forward all of the other table spaces in which that table resides to the same point in time. Roll-forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached or detached data partitions, then PIT rollforward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it will generally be necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory specified in the ROLLFORWARD DATABASE command. It is possible to roll forward all table spaces in one command, or do repeated roll forward operations for subsets of the table spaces involved. If the ROLLFORWARD DATABASE command is done for one or a few table spaces, then all data from the table that resided in those table spaces will be recovered. A warning will be written to the notify log if the ROLLFORWARD DATABASE command did not specify the full set of the table spaces necessary to recover all the data for the table. Allowing rollforward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

ONLINE
This keyword is specified to allow table space-level rollforward recovery to be done online. This means that other agents are allowed to connect while rollforward recovery is in progress.

OVERFLOW LOG PATH log-directory
Specifies an alternate log path to be searched for archived logs during recovery. Use this parameter if log files were moved to a location other than that specified by the *logpath* database configuration parameter. In a partitioned database environment, this is the (fully qualified) default overflow log path *for all database partitions*. A relative overflow log path can be specified for single-partition databases. The OVERFLOW LOG PATH command parameter will overwrite the value (if any) of the database configuration parameter OVERFLOWLOGPATH.

log-directory ON DBPARTITIONNUM
In a partitioned database environment, allows a different log path to override the default overflow log path for a specific database partition.

NORETRIEVE
Allows you to control which log files are to be rolled forward on the standby machine by allowing you to disable the retrieval of archived logs. The benefits of this are:

- By controlling the logfiles to be rolled forward, you can ensure that the standby machine is X hours behind the production machine, to avoid affecting both the systems.
- If the standby system does not have access to archive (eg. if TSM is the archive, it only allows the original machine to retrieve the files)
- It might also be possible that while the production system is archiving a file, the standby system is retrieving the same file, and it might then get an incomplete log file. Noretrieve would solve this problem.

RECOVER DROPPED TABLE drop-table-id
Recovers a dropped table during the rollforward operation. The table ID can be obtained using the LIST HISTORY command. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward command.

TO export-directory
Specifies a directory to which files containing the table data are to be written. The directory must be accessible to all database partitions.

Examples:

Example 1

The ROLLFORWARD DATABASE command permits specification of multiple operations at once, each being separated with the keyword AND. For example, to roll forward to the end of logs, and complete, the separate commands:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

can be combined as follows:

```
db2 rollforward db sample to end of logs and complete
```

Although the two are equivalent, it is recommended that such operations be done in two steps. It is important to verify that the rollforward operation has progressed as expected, before stopping it and possibly missing logs. This is especially important if a bad log is found during rollforward recovery, and the bad log is interpreted to mean the "end of logs". In such cases, an undamaged backup copy of that log could be used to continue the rollforward operation through more logs. However if the rollforward AND STOP option is used, and the rollforward encounters an error, the error will be returned to you. In this case, the only way to force the rollforward to stop and come online despite the error (i.e. to come online at that point in the logs before the error) is to issue the rollforward STOP command.

Example 2

Roll forward to the end of the logs (two table spaces have been restored):

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

These two statements are equivalent. Neither AND STOP or AND COMPLETE is needed for table space rollforward recovery to the end of the logs. Table space names are not required. If not specified, all table spaces requiring rollforward recovery will be included. If only a subset of these table spaces is to be rolled forward, their names must be specified.

Example 3

After three table spaces have been restored, roll one forward to the end of the logs, and the other two to a point in time, both to be done online:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS2, TBS3) online
```

Two rollforward operations cannot be run concurrently. The second command can only be invoked after the first rollforward operation completes successfully.

Example 4

After restoring the database, roll forward to a point in time, using OVERFLOW LOG PATH to specify the directory where the user exit saves archived logs:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   overflow log path (/logs)
```

Example 5 (partitioned database environments)

There are three database partitions: 0, 1, and 2. Table space TBS1 is defined on all database partitions, and table space TBS2 is defined on database partitions 0 and 2. After restoring the database on database partition 1, and TBS1 on database partitions 0 and 2, roll the database forward on database partition 1:

```
db2 rollforward db sample to end of logs and stop
```

This returns warning SQL1271 ("Database is recovered but one or more table spaces are off-line on database partition(s) 0 and 2.").

```
db2 rollforward db sample to end of logs
```

This rolls TBS1 forward on database partitions 0 and 2. The clause TABLESPACE(TBS1) is optional in this case.

Example 6 (partitioned database environments)

After restoring table space TBS1 on database partitions 0 and 2 only, roll TBS1 forward on database partitions 0 and 2:

```
db2 rollforward db sample to end of logs
```

Database partition 1 is ignored.

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1. Reports SQL4906N.

```
db2 rollforward db sample to end of logs on dbpartitionnums (0, 2)
   tablespace(TBS1)
```

This completes successfully.

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS1)
```

This fails, because TBS1 is not ready for rollforward recovery on database partition 1; all pieces must be rolled forward together. With table space rollforward to a point in time, the database partition clause is not accepted. The rollforward operation must take place on all the database partitions on which the table space resides.

After restoring TBS1 on database partition 1:

```
db2 rollforward db sample to 1998-04-03-14.21.56 and stop
   tablespace(TBS1)
```

This completes successfully.

Example 7 (partitioned database environment)

After restoring a table space on all database partitions, roll forward to point in time 2, but do not specify `AND STOP`. The rollforward operation is still in progress. Cancel and roll forward to point in time 1:

```
   db2 rollforward db sample to pit2 tablespace(TBS1)
   db2 rollforward db sample cancel tablespace(TBS1)

 ** restore TBS1 on all database partitions **

   db2 rollforward db sample to pit1 tablespace(TBS1)
   db2 rollforward db sample stop tablespace(TBS1)
```

Example 8 (partitioned database environments)

Rollforward recover a table space that resides on eight database partitions (3 to 10) listed in the db2nodes.cfg file:

```
   db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

This operation to the end of logs (not point in time) completes successfully. The database partitions on which the table space resides do not have to be specified. The utility defaults to the db2nodes.cfg file.

Example 9 (partitioned database environment)

Rollforward recover six small table spaces that reside on a single-partition database partition group (on database partition 6):

```
   db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
      tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

This operation to the end of logs (not point in time) completes successfully.

Usage notes:

If restoring from an image that was created during an online backup operation, the specified point in time for the rollforward operation must be later than the time at which the online backup operation completed. If the rollforward operation is stopped before it passes this point, the database is left in rollforward pending state. If a table space is in the process of being rolled forward, it is left in rollforward in progress state.

If one or more table spaces is being rolled forward to a point in time, the rollforward operation must continue at least to the minimum recovery time, which is the last update to the system catalogs for this table space or its tables. The minimum recovery time (in Coordinated Universal Time, or UTC) for a table space can be retrieved using the LIST TABLESPACES SHOW DETAIL command.

Rolling databases forward might require a load recovery using tape devices. If prompted for another tape, you can respond with one of the following:

c	Continue. Continue using the device that generated the warning message (for example, when a new tape has been mounted)
d	Device terminate. Stop using the device that generated the warning message (for example, when there are no more tapes)
t	Terminate. Take all affected tablespaces offline, but continue rollforward processing.

If the rollforward utility cannot find the next log that it needs, the log name is returned in the SQLCA, and rollforward recovery stops. If no more logs are available, use the STOP option to terminate rollforward recovery. Incomplete transactions are rolled back to ensure that the database or table space is left in a consistent state.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODE can be substituted for DBPARTITIONNUM.
- The keyword NODES can be substituted for DBPARTITIONNUMS.
- Point in time rollforward is not supported with pre-V9.1 clients due to V9.1 support for partitioned tables.

Related concepts:
- "Developing a backup and recovery strategy" in *Data Recovery and High Availability Guide and Reference*

Related tasks:
- "Using rollforward" in *Data Recovery and High Availability Guide and Reference*

SET RUNTIME DEGREE

Sets the maximum run time degree of intra-partition parallelism for SQL statements for specified active applications.

Scope:

This command affects all database partitions that are listed in the `$HOME/sqllib/db2nodes.cfg` file.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*

Required connection:

Instance. To change the maximum run time degree of intra-partition parallelism on a remote server, it is first necessary to attach to that server. If no attachment exists, the SET RUNTIME DEGREE command fails.

Command syntax:

```
>>-SET RUNTIME DEGREE FOR--+-ALL--------------------------+--TO--degree-------><
                           |    .-,------------------.    |
                           |    V                    |    |
                           '-(----application-handle-+--)-'
```

Command parameters:

FOR

ALL The specified degree will apply to all applications.

application-handle
Specifies the agent to which the new degree applies. List the values using the LIST APPLICATIONS command.

TO degree
The maximum run time degree of intra-partition parallelism.

Examples:

The following example sets the maximum run time degree of parallelism for two users, with *application-handle* values of 41408 and 55458, to 4:

```
db2 SET RUNTIME DEGREE FOR ( 41408, 55458 ) TO 4
```

Usage notes:

This command provides a mechanism to modify the maximum degree of parallelism for active applications. It can be used to override the value that was determined at SQL statement compilation time.

The run time degree of intra-partition parallelism specifies the maximum number of parallel operations that will be used when the statement is executed. The degree of intra-partition parallelism for an SQL statement can be specified at statement compilation time using the CURRENT DEGREE special register or the **degree** bind option. The maximum run time degree of intra-partition parallelism for an active application can be specified using the SET RUNTIME DEGREE command. The *max_querydegree* database manager configuration parameter specifies the maximum run time degree for any SQL statement executing on this instance of the database manager.

The actual run time degree will be the lowest of:
- the *max_querydegree* configuration parameter
- the application run time degree
- the SQL statement compilation degree.

Related tasks:
- "Enabling intra-partition parallelism for queries" in *Administration Guide: Implementation*

Related reference:
- "sqlesdeg - Set the maximum runtime intra-partition parallelism level or degree for SQL statements" on page 816
- "LIST APPLICATIONS " on page 530

SET WRITE

The SET WRITE command allows a user to suspend I/O writes or to resume I/O writes for a database. Typical use of this command is for splitting a mirrored database. This type of mirroring is achieved through a disk storage system.

This new state, SUSPEND_WRITE, is visible from the Snapshot Monitor. All table spaces must be in a NORMAL state for the command to execute successfully. If any one table space is in a state other than NORMAL, the command will fail.

Scope:

This command only affects the database partition on which it is executed.

Authorization:

This command only affect the node on which it is executed. The authorization of this command requires the issuer to have one of the following privileges:
- sysadm
- sysctrl
- sysmaint

Required Connection:

Database

Command Syntax:

```
►►──SET──WRITE──┬─SUSPEND─┬──FOR──┬─DATABASE─┬──────────────────────────────►◄
                └─RESUME──┘       └─DB───────┘
```

Command Parameters:

SUSPEND
: Suspending I/O writes will put all table spaces into a new state SUSPEND_WRITE state. Writes to the logs are also suspended by this command. All database operations, apart from online backup and restore, should function normally while database writes are suspended. However, some operations can wait while attempting to flush dirty pages from the buffer pool or log buffers to the logs. These operations will resume normally once the database writes are resumed.

RESUME
: Resuming I/O writes will remove the SUSPEND_WRITE state from all of the table spaces and make the table spaces available for update.

Usage Notes:

It is suggested that I/O writes be resumed from the same connection from which they were suspended. Ensuring that this connection is available to resume I/O writes involves not performing any operations from this connection until database writes are resumed. Otherwise, some operations can wait for I/O writes to be resumed if dirty pages must be flushed from the buffer pool or from log buffers to the logs. Furthermore, subsequent connection attempts might hang if they require flushing dirty pages from the buffer pool to disk. Subsequent connections will complete successfully once database I/O resumes. If your connection attempts are hanging, and it has become impossible to resume I/O from the connection that you used to suspend I/O, then you will have to run the RESTART DATABASE command with the WRITE RESUME option. When used in this circumstance, the RESTART DATABASE command will resume I/O writes without performing crash recovery. The RESTART DATABASE command with the WRITE RESUME option will only perform crash recovery when you use it after a database crash.

Related concepts:

- "High availability through online split mirror and suspended I/O support" in *Data Recovery and High Availability Guide and Reference*

Related tasks:

- “Using a split mirror as a backup image” in *Data Recovery and High Availability Guide and Reference*
- “Using a split mirror as a standby database” in *Data Recovery and High Availability Guide and Reference*
- “Using a split mirror to clone a database” in *Data Recovery and High Availability Guide and Reference*

Related reference:

- “db2SetWriteForDB - Suspend or resume I/O writes for database” on page 777

START DATABASE MANAGER

Starts the current database manager instance background processes on a single database partition or on all the database partitions defined in a multi-partitioned database environment.

Scope:

In a multi-partitioned database environment, this command affects all database partitions that are listed in the `$HOME/sqllib/db2nodes.cfg` file, unless the *dbpartitionnum* parameter is used.

Authorization:

One of the following:

- *sysadm*
- *sysctrl*
- *sysmaint*

The ADD DBPARTITIONNUM start option requires either *sysadm* or *sysctrl* authority.

You must meet Windows operating system requirements for starting a service by being a member of the Administrators, Server Operators or Power Users group. With Extended Security enabled, you must also be a member of either the Administrators group or the DB2ADMINS group to start the database.

Required connection:

None

Command syntax:

```
►►──┬─START──┬─DATABASE MANAGER─┬──┬────────────────────────────────────────────►
    │        ├─DB MANAGER───────┤  │
    │        └─DBM──────────────┘  │
    └─db2start──┬────┬─────────────┘
                └─/D─┘

►──┬──────────────────────────────────────────────────────────────┬─────────────►
   └─REMOTE──┬──────────┬──instancename──┤ remote options ├───────┘
             └─INSTANCE─┘
```

```
►─┬──────────────────────────────────┬──┬────────────────────┬──────────────────────────►
  └─ADMIN MODE─┬──────────────────┬──┘  └─PROFILE──profile─┘
               ├─USER──username───┤
               └─GROUP──groupname─┘

►─┬───────────────────────────────────────────────────────────┬──────────────────────►◄
  └─DBPARTITIONNUM──db-partition-number──┤ start options ├────┘
```

remote options:

```
├──┬─ADMINNODE──nodename─┬──USER──username──USING──password──────────────────────────┤
   └─HOSTNAME──hostname──┘
```

start options:

```
├──┬─────────────────────────────────────────────────────────────┬────────────────────┤
   ├─ADD DBPARTITIONNUM──┤ add dbpartitionnum options ├──────────┤
   ├─STANDALONE──────────────────────────────────────────────────┤
   └─RESTART──┤ restart options ├────────────────────────────────┘
```

add dbpartitionnum options:

```
├──HOSTNAME──hostname──PORT──logical-port──┬──────────────────────────────┬─────────────►
                                           └─COMPUTER──computer-name──────┘

►─┬──────────────────┬──┬──────────────────────┬──┬────────────────────┬────────────────►
  └─USER──username───┘  └─PASSWORD──password───┘  └─NETNAME──netname───┘

►─┬─────────────────────────────────────────────┬───────────────────────────────────────┤
  ├─LIKE DBPARTITIONNUM──db-partition-number────┤
  └─WITHOUT TABLESPACES─────────────────────────┘
```

restart options:

```
├──┬──────────────────────┬──┬─────────────────────┬──┬──────────────────────────┬──────►
   └─HOSTNAME──hostname───┘  └─PORT──logical-port──┘  └─COMPUTER──computername───┘

►─┬──────────────────┬──┬──────────────────────┬──┬────────────────────┬────────────────┤
  └─USER──username───┘  └─PASSWORD──password───┘  └─NETNAME──netname───┘
```

Command parameters:

REMOTE [INSTANCE] instancename
Specifies the name of the remote instance you wish to start.

ADMINNODE nodename
With REMOTE, or REMOTE INSTANCE, specifies the name of the administration node.

HOSTNAME hostname
With REMOTE, or REMOTE INSTANCE, specifies the name of the host node.

USER username
With REMOTE, or REMOTE INSTANCE, specifies the name of the user.

USING password
With REMOTE, or REMOTE INSTANCE, and the USER, specifies the password of the user.

ADMIN MODE
Starts the instance in quiesced mode for administration purposes. This is equivalent to the QUIESCE INSTANCE command except in this case the instance is not already "up", and therefore there is no need to force the connections off.

USER username
With ADMIN MODE, specifies the name of the user.

GROUP groupname
With ADMIN MODE, specifies the name of the group.

All of the following parameters are valid in an Enterprise Server Edition (ESE) environment only.

PROFILE profile
Specifies the name of the profile file to be executed at each database partition to define the DB2 environment. This file is executed before the database partitions are started. The profile file must reside in the `sqllib` directory of the instance owner. The environment variables in the profile file are not necessarily all defined in the user session.

DBPARTITIONNUM db-partition-number
Specifies the database partition to be started. If no other options are specified, a normal startup is done at this database partition.

Valid values are from 0 to 999 inclusive. If ADD DBPARTITIONNUM is not specified, the value must already exist in the `db2nodes.cfg` file of the instance owner. If no database partition number is specified, all database partitions defined in the configuration file are started.

ADD DBPARTITIONNUM
Specifies that the new database partition is added to the `db2nodes.cfg` file of the instance owner with the *hostname* and *logical-port* values.

Ensure that the combination of *hostname* and *logical-port* is unique.

The add database partition utility is executed internally to create all existing databases on the database partition being added. After a database partition is added, the `db2nodes.cfg` file is not updated with the new database partition until a **db2stop** is issued. The database partition is not part of the MPP system until the next **db2start** following the **db2stop**.

When the database partitions are created on the new node, their configuration parameters are set to the default.

HOSTNAME hostname
With ADD DBPARTITIONNUM, specifies the host name to be added to the `db2nodes.cfg` file.

PORT logical-port
With ADD DBPARTITIONNUM, specifies the logical port to be added to the `db2nodes.cfg` file. Valid values are from 0 to 999.

COMPUTER computername
The computer name for the machine on which the new database partition is created. This parameter is mandatory on Windows, but is ignored on other operating systems.

USER username
The user name for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

PASSWORD password
The password for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

NETNAME netname
Specifies the *netname* to be added to the `db2nodes.cfg` file. If not specified, this parameter defaults to the value specified for *hostname*.

LIKE DBPARTITIONNUM db-partition-number
Specifies that the containers for the system temporary table spaces will be the same as the containers on the specified *db-partition-number* for each database in the instance. The database partition specified must be a database partition that is already in the `db2nodes.cfg` file. For system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement or where no MANAGED BY CLAUSE was specified at all), the containers will not necessarily match those from the partition specified. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. This may or may not result in the same containers being used on these two partitions.

WITHOUT TABLESPACES
Specifies that containers for the system temporary table spaces are not created for any of the databases. The ALTER TABLESPACE statement must be used to add system temporary table space containers to each database before the database can be used. This option is ignored for system temporary table spaces that are defined to use automatic storage (in other words, system temporary table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement or where no MANAGED BY CLAUSE was specified at all). For these table spaces, there is no way to defer container creation. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database.

STANDALONE
Specifies that the database partition is to be started in stand-alone mode. FCM does not attempt to establish a connection to any other database partition. This option is used when adding a database partition.

RESTART
Starts the database manager after a failure. Other database partitions are still operating, and this database partition attempts to connect to the others. If neither the *hostname* nor the *logical-port* parameter is specified, the database manager is restarted using the *hostname* and *logical-port* values specified in `db2nodes.cfg`. If either parameter is specified, the new values are sent to the other database partitions when a connection is established. The `db2nodes.cfg` file is updated with this information.

HOSTNAME hostname
With RESTART, specifies the host name to be used to override that in the database partition configuration file.

PORT logical-port
With RESTART, specifies the logical port number to be used to override that in the database partition configuration file. If not specified, this parameter defaults to the *logical-port* value that corresponds to the *num* value in the `db2nodes.cfg` file. Valid values are from 0 to 999.

COMPUTER computername
The computer name for the machine on which the new database partition is created. This parameter is mandatory on Windows, but is ignored on other operating systems.

USER username
The user name for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

PASSWORD password
The password for the account on the new database partition. This parameter is mandatory on Windows, but is ignored on other operating systems.

NETNAME netname
Specifies the *netname* to override that specified in the `db2nodes.cfg` file. If not specified, this parameter defaults to the *netname* value that corresponds to the *db-partition-number* value in the `db2nodes.cfg` file.

Examples:

The following is sample output from **db2start** issued on a three-database partition system with database partitions 10, 20, and 30:

```
04-07-1997 10:33:05    10   0   SQL1063N  DB2START processing was successful.
04-07-1997 10:33:07    20   0   SQL1063N  DB2START processing was successful.
04-07-1997 10:33:07    30   0   SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
```

Usage notes:

It is not necessary to issue this command on a client node. It is provided for compatibility with older clients, but it has no effect on the database manager.

Once started, the database manager instance runs until the user stops it, even if all application programs that were using it have ended.

If the database manager starts successfully, a successful completion message is sent to the standard output device. If an error occurs, processing stops, and an error message is sent to the standard output device. In a partitioned database environment, messages are returned on the database partition that issued the START DATABASE MANAGER command.

If no parameters are specified in a partitioned database environment, the database manager is started on all parallel nodes using the parameters specified in the database partition configuration file.

If a START DATABASE MANAGER command is in progress, ensure that the applicable database partitions have started *before* issuing a request to the database.

The `db2cshrc` file is not supported and cannot be used to define the environment.

You can start an instance in a quiesced state. You can do this by using one of the following choices:

```
db2start admin mode
```

or

```
db2start admin mode user username
```

or

```
db2start admin mode group groupname
```

When adding a new database partition, `START DATABASE MANAGER` must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for each database. If automatic storage is enabled then the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, START DATABASE MANAGER might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. The start_stop_time database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the command fails. If this situation occurs, increase the value of `start_stop_time`, and reissue the command.

On UNIX platforms, the START DATABASE MANAGER command supports the SIGINT signal. It is issued if CTRL+C is pressed. If this signal occurs, all in-progress startups are interrupted and a message (SQL1044N) is returned from each interrupted database partition to the `$HOME/sqllib/log/db2start.`*timestamp*`.log` error log file. Database partitions that are already started are not affected. If CTRL+C is pressed on a database partition that is starting, **db2stop** must be issued on that database partition before an attempt is made to start it again.

On Windows operating systems, neither the `db2start` command nor the `NET START` command returns warnings if any communication subsystem failed to start. The database manager in a Windows environment is implemented as a service, and does not return an error if the service is started successfully. Be sure to examine the Event Log or the `DB2DIAG.LOG` file for any errors that might have occurred during the running of `db2start`.

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keywords LIKE NODE can be substituted for LIKE DBPARTITIONNUM.
- The keyword ADDNODE can be substituted for ADD DBPARTITIONNUM.
- The keyword NODENUM can be substituted for DBPARTITIONNUM.

Related reference:
- "STOP DATABASE MANAGER " on page 624

- "ADD DBPARTITIONNUM " on page 435
- "db2InstanceStart - Start instance" on page 714

STOP DATABASE MANAGER

Stops the current database manager instance. Unless explicitly stopped, the database manager continues to be active. This command does not stop the database manager instance if any applications are connected to databases. If there are no database connections, but there are instance attachments, it forces the instance attachments and stops the database manager. This command also deactivates any outstanding database activations before stopping the database manager.

On partitioned database system, this command stops the current database manager instance on a database partition or on all database partitions. When it stops the database manager on all database partitions, it uses the `db2nodes.cfg` configuration file to obtain information about each database partition.

This command can also be used to drop a database partition from the `db2nodes.cfg` file (partitioned database systems only).

This command is not valid on a client.

Scope:

By default, and in a partitioned database environment, this command affects all database partitions that are listed in the `db2nodes.cfg` file.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

None

Command syntax:

```
>>──┬─STOP──┬─DATABASE MANAGER─┬─┬──┬─────────────────────┬──────────────>
    │       ├─DB MANAGER───────┤ │  └─PROFILE──profile─┘
    │       └─DBM──────────────┘ │
    └─db2stop────────────────────┘

>──┬──────────────────────────────────────────────┬─────────────────────><
   ├─DBPARTITIONNUM──db-partition-number──────────┤
   ├─DROP DBPARTITIONNUM──db-partition-number─────┤
   └─FORCE──┬──────────────────────────────────┬──┘
            └─DBPARTITIONNUM──db-partition-number─┘
```

Command parameters:

PROFILE profile
: partitioned database systems only. Specifies the name of the profile file that was executed at startup to define the DB2 environment for those database partitions that were started. If a profile for the START DATABASE MANAGER command was specified, the same profile must be specified here. The profile file must reside in the `sqllib` directory of the instance owner.

DBPARTITIONNUM db-partition-number
: partitioned database systems only. Specifies the database partition to be stopped.

  Valid values are from 0 to 999 inclusive, and must be in the `db2nodes.cfg` file. If no database partition number is specified, all database partitions defined in the configuration file are stopped.

DROP DBPARTITIONNUM db-partition-number
: partitioned database systems only. Specifies the database partition to be dropped from the `db2nodes.cfg` file.

  Before using this parameter, run the DROP DBPARTITIONNUM VERIFY command to ensure that there is no user data on this database partition.

  When this option is specified, all database partitions in the `db2nodes.cfg` file are stopped.

FORCE
: Specifies to use FORCE APPLICATION ALL when stopping the database manager at each database partition.

DBPARTITIONNUM db-partition-number
: partitioned database systems only. Specifies the database partition to be stopped after all applications on that database partition have been forced to stop. If the FORCE option is used without this parameter, all applications on all database partitions are forced before all the database partitions are stopped.

Examples:

The following is sample output from **db2stop** issued on a three-partition system with database partitions 10, 20, and 30:

```
04-07-1997 10:32:53    10   0   SQL1064N  DB2STOP processing was successful.
04-07-1997 10:32:54    20   0   SQL1064N  DB2STOP processing was successful.
04-07-1997 10:32:55    30   0   SQL1064N  DB2STOP processing was successful.
SQL1064N  DB2STOP processing was successful.
```

Usage notes:

It is not necessary to issue this command on a client node. It is provided for compatibility with older clients, but it has no effect on the database manager.

Once started, the database manager instance runs until the user stops it, even if all application programs that were using it have ended.

If the database manager is stopped, a successful completion message is sent to the standard output device. If an error occurs, processing stops, and an error message is sent to the standard output device.

If the database manager cannot be stopped because application programs are still connected to databases, use the FORCE APPLICATION command to disconnect all users first, or reissue the STOP DATABASE MANAGER command with the FORCE option.

The following information applies to partitioned database environments only:
- If no parameters are specified, the database manager is stopped on each database partition listed in the configuration file. The administration notification log might contain messages to indicate that other database partitions are shutting down.
- Any database partitions added to the partitioned database system since the previous STOP DATABASE MANAGER command was issued will be updated in the `db2nodes.cfg` file.
- On UNIX platforms, if the value specified for the *start_stop_time* database manager configuration parameter is reached, all in-progress stops are interrupted, and message SQL6037N is returned from each interrupted database partition to the `$HOME/sqllib/log/db2stop.` *timestamp*`.log` error log file. Database partitions that are already stopped are not affected.
- The `db2cshrc` file is not supported and cannot be specified as the value for the PROFILE parameter.

Attention: The UNIX **kill** command should *not* be used to terminate the database manager because it will abruptly end database manager processes without controlled termination and cleanup processing.

Related reference:
- "FORCE APPLICATION command" in *Command Reference*
- "START DATABASE MANAGER " on page 618
- "DEACTIVATE DATABASE command" in *Command Reference*
- "DROP DBPARTITIONNUM VERIFY " on page 495
- "db2InstanceStop - Stop instance" on page 719

UNQUIESCE

Unless specifically designated, no user except those with *sysadm*, *sysmaint*, or *sysctrl* has access to a database while it is quiesced. Therefore an UNQUIESCE is required to restore general access to a quiesced database.

Scope:

UNQUIESCE DB restores user access to all objects in the quiesced database.

To stop the instance and unquiesce it and all its databases, issue the `db2stop` command. Stopping and restarting DB2 will unquiesce all instances and databases.

Authorization:

One of the following:

For database level unquiesce:
- *sysadm*
- *dbadm*

Command syntax:

```
►►──UNQUIESCE──DB──────────────────────────────────────────────────────►◄
```

Required connection:
Database
Command parameters:

DB Unquiesce the database. User access will be restored to all objects in the database.

Examples:

Unquiescing a Database

This command will unquiesce the database that had previously been quiesced.

Related reference:
- "QUIESCE " on page 567
- "db2DatabaseUnquiesce - Unquiesce database" on page 683
- "db2InstanceUnquiesce - Unquiesce instance" on page 722
- "UNQUIESCE DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

UPDATE DATABASE CONFIGURATION

Modifies individual entries in a specific database configuration file.

A database configuration file resides on every database partition on which the database has been created.

Scope:

This command only affects the database partition on which it is executed.

Authorization:

One of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

Required connection:

Command syntax:

```
►►──UPDATE──┬─DATABASE─┬──┬─CONFIGURATION─┬──┬────────────────────────┬──────────►
            └─DB───────┘  ├─CONFIG────────┤  └─FOR──database-alias─┘
                          └─CFG───────────┘
```

```
        ┌──────────────────────┐
        ▼                      │ ┌─IMMEDIATE─┐
►─USING───config-keyword value─┴─┼───────────┼──────────────────────────►◄
                                 └─DEFERRED──┘
```

Command parameters:

DEFERRED
Make the changes only in the configuration file, so that the changes take effect the next time you reactivate the database.

FOR database-alias
Specifies the alias of the database whose configuration is to be updated. Specifying the database alias is not required when a database connection has already been established. You can update the configuration file for another database residing under the same database instance. For example, if you are connected only to database `db11`, and issue `update db config for alias db22 using .... immediate`:

- If there is no active connection on `db22`, the update will be successful because only the configuration file needs to be updated. A new connection (which will activate the database) will see the new change in memory.
- If there are active connections on `db22` from other applications, the update will work on disk but not in memory. You will receive a warning saying that the database needs to be restarted.

IMMEDIATE
Make the changes immediately, while the database is running.

USING *config-keyword value*
config-keyword specifies the database configuration parameter to be updated. *value* specifies the value to be assigned to the parameter.

Usage notes:

For more information about DB2 configuration parameters and the values available for each type of database node, see the individual configuration parameter descriptions. The values of these parameters differ for each type of database node configured (server, client, or server with remote clients).

Not all parameters can be updated.

Some changes to the database configuration file become effective only after they are loaded into memory. All applications must disconnect from the database before this can occur. For more information on which parameters are configurable on-line and which ones are not, see summary list of configuration parameters.

If an error occurs, the database configuration file does not change. The database configuration file cannot be updated if the checksum is invalid. This might occur if the database configuration file is changed without using the appropriate command. If this happens, the database must be restored to reset the database configuration file.

Related concepts:

- "rah and db2_all commands overview" on page 417

Related tasks:

- “Configuring DB2 with configuration parameters” on page 1493

Related reference:

- “db2CfgSet - Set the database manager or database configuration parameters” on page 676
- “Configuration parameters summary” on page 1484
- “GET DATABASE CONFIGURATION ” on page 502
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “DBCFG administrative view - Retrieve database configuration parameter information” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

UPDATE DATABASE MANAGER CONFIGURATION

Modifies individual entries in the database manager configuration file.

Authorization:

sysadm

Required connection:

Command syntax:

```
►►──UPDATE──┬─DATABASE MANAGER─┬──┬─CONFIGURATION─┬──────────────────────────►
            ├─DB MANAGER───────┤  ├─CONFIG────────┤
            └─DBM──────────────┘  └─CFG───────────┘

            ┌──────────────────────┐
            ▼                      │  ┌─IMMEDIATE─┐
►─USING─────config-keyword value───┴──┼───────────┼──────────────────────────►◄
                                      └─DEFERRED──┘
```

Command parameters:

DEFERRED
: Make the changes only in the configuration file, so that the changes take effect when the instance is restarted.

IMMEDIATE
: Make the changes right now, dynamically, while the instance is running.

USING config-keyword value
: Specifies the database manager configuration parameter to be updated. For a list of configuration parameters, refer to the configuration parameters summary.

Usage notes:

Not all parameters can be updated.

Some changes to the database manager configuration file become effective only after they are loaded into memory. For more information on which parameters are configurable on-line and which ones are not, see the configuration parameter summary. Server configuration parameters that are not reset immediately are reset during execution of **db2start**. For a client configuration parameter, parameters are reset the next time you restart the application. If the client is the command line processor, it is necessary to invoke TERMINATE.

If an error occurs, the database manager configuration file does not change.

The database manager configuration file cannot be updated if the checksum is invalid. This can occur if you edit database manager configuration file and do not use the appropriate command. If the checksum is invalid, you must reinstall the database manager to reset the database manager configuration file.

When you update the SVCENAME, or TPNAME database manager configuration parameters for the current instance, if LDAP support is enabled and there is an LDAP server registered for this instance, the LDAP server is updated with the new value or values.

Related tasks:

- “Configuring DB2 with configuration parameters” on page 1493

Related reference:

- “db2CfgSet - Set the database manager or database configuration parameters” on page 676
- “Configuration parameters summary” on page 1484
- “GET DATABASE MANAGER CONFIGURATION ” on page 508
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “TERMINATE command” in *Command Reference*
- “DBMCFG administrative view - Retrieve database manager configuration parameter information” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

Chapter 37. Commands for Users

ATTACH

Enables an application to specify the instance at which instance-level commands (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This instance can be the current instance, another instance on the same workstation, or an instance on a remote workstation.

Authorization:

None

Required connection:

None. This command establishes an instance attachment.

Command syntax:

```
>>--ATTACH--+----------------+------------------------------------------>
            '-TO--nodename---'

>--+--------------------------------------------------------------+------><
   '-USER--username--+-------------------------------------------+-'
                     +-USING--password--+---------------------------------+-+
                     |                  '-NEW--password--CONFIRM--password-' |
                     '-CHANGE PASSWORD---------------------------------------'
```

Command parameters:

TO nodename
: Alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception to this is the local instance (as specified by the **DB2INSTANCE** environment variable) which can be specified as the object of an attach, but which cannot be used as a node name in the node directory.

USER username
: Specifies the authentication identifier. When attaching to a DB2 database instance on a Windows operating system, the user name can be specified in a format compatible with Microsoft Security Account Manager (SAM). The qualifier must be a NetBIOS-style name, which has a maximum length of 15 characters. For example, *domainname\username*.

USING password
: Specifies the password for the user name. If a user name is specified, but a password is *not* specified, the user is prompted for the current password. The password is not displayed at entry.

NEW password
: Specifies the new password that is to be assigned to the user name.

Passwords can be up to 18 characters in length. The system on which the password will be changed depends on how user authentication has been set up.

CONFIRM password
A string that must be identical to the new password. This parameter is used to catch entry errors.

CHANGE PASSWORD
If this option is specified, the user is prompted for the current password, a new password, and for confirmation of the new password. Passwords are not displayed at entry.

Examples:

Catalog two remote nodes:

```
db2 catalog tcpip node node1 remote freedom server server1
db2 catalog tcpip node node2 remote flash server server1
```

Attach to the first node, force all users, and then detach:

```
db2 attach to node1
db2 force application all
db2 detach
```

Attach to the second node, and see who is on:

```
db2 attach to node2
db2 list applications
```

After the command returns agent IDs 1, 2 and 3, force 1 and 3, and then detach:

```
db2 force application (1, 3)
db2 detach
```

Attach to the current instance (not necessary, will be implicit), force all users, then detach (AIX only):

```
db2 attach to $DB2INSTANCE
db2 force application all
db2 detach
```

Usage notes:

If *nodename* is omitted from the command, information about the current state of attachment is returned.

If ATTACH has not been executed, instance-level commands are executed against the current instance, specified by the **DB2INSTANCE** environment variable.

Related tasks:

- "Attaching to and detaching from a non-default instance of the database manager" in *Administration Guide: Implementation*

Related reference:

- "DETACH " on page 633
- "sqleatcp - Attach to instance and change password" on page 836
- "sqleatin - Attach to instance" on page 838

DETACH

Removes the logical DBMS instance attachment, and terminates the physical communication connection if there are no other logical connections using this layer.

Authorization:

None

Required connection:

None. Removes an existing instance attachment.

Command syntax:

```
►►──DETACH──────────────────────────────────────────────────────►◄
```

Command parameters:
None
Related tasks:
- "Attaching to and detaching from a non-default instance of the database manager" in *Administration Guide: Implementation*

Related reference:
- "ATTACH " on page 631
- "sqledtin - Detach from instance" on page 841

GET CONNECTION STATE

Displays the connection state. Possible states are:
- Connectable and connected
- Connectable and unconnected
- Unconnectable and connected
- Implicitly connectable (if implicit connect is available).

This command also returns information about:
- the database connection mode (SHARE or EXCLUSIVE)
- the alias and name of the database to which a connection exists (if one exists)
- the host name and service name of the connection if the connection is using TCP/IP

Authorization:

None

Required connection:

None

Command syntax:

```
►►──GET CONNECTION STATE──────────────────────────────────────────►◄
```

Command parameters:
None
Examples:
The following is sample output from GET CONNECTION STATE:

```
   Database Connection State

 Connection state       = Connectable and Connected
 Connection mode        = SHARE
 Local database alias   = SAMPLE
 Database name          = SAMPLE
 Hostname               = montero
 Service name           = 29384
```

Usage notes:

This command does not apply to type 2 connections.

Related reference:
- "SET CLIENT command" in *Command Reference*
- "UPDATE ALTERNATE SERVER FOR DATABASE command" in *Command Reference*

LIST DBPARTITIONNUMS

Lists all database partitions associated with the current database.

Scope:

This command can be issued from any database partition that is listed in `$HOME/sqllib/db2nodes.cfg`. It returns the same information from any of these database partitions.

Authorization:

None

Required connection:

Database

Command syntax:

```
►►──LIST DBPARTITIONNUMS──────────────────────────────────────────►◄
```

Command parameters:
None
Examples:
Following is sample output from the LIST DBPARTITIONNUMS command:

```
DATABASE PARTITION NUMBER
-------------------------
                        0
                        2
                        5
                        7
                        9

     5 record(s) selected.
```

Compatibilities:

For compatibility with versions earlier than Version 8:
- The keyword NODES can be substituted for DBPARTITIONNUMS.

Related reference:
- "REDISTRIBUTE DATABASE PARTITION GROUP " on page 577

PRECOMPILE

Processes an application program source file containing embedded SQL statements. A modified source file is produced, containing host language calls for the SQL statements and, by default, a package is created in the database.

Scope:

This command can be issued from any database partition in `db2nodes.cfg`. In a partitioned database environment, it can be issued from any database partition server defined in the db2nodes.cfg file. It updates the database catalogs on the catalog database partition. Its effects are visible to all database partitions.

Authorization:

One of the following:
- *sysadm* or *dbadm* authority
- BINDADD privilege if a package does not exist, and one of:
  - IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
  - CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has *sysadm* authority, but not explicit privileges to complete the bind, the database manager grants explicit *dbadm* authority automatically.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

Command syntax:

PRECOMPILE

For DB2 for Windows and UNIX

```
►►──┬─PRECOMPILE─┬──filename─────────────────────────────────────────────►
    └─PREP───────┘

►──┬──────────────────────────────────────────────────────────────────┬──►
   └─ACTION──┬─ADD────────────────────────────────────────────────────┤
             │ ┌─REPLACE─┐                                            │
             └─┴─────────┴──┬─────────────────┬──┬─────────────────────┬┘
                            └─RETAIN──┬─NO──┬─┘  └─REPLVER──version-id─┘
                                      └─YES─┘

►──┬────────────────────────────┬──┬─────────────────────────┬───────────►
   └─BINDFILE─┬───────────────┬─┘  └─BLOCKING──┬─UNAMBIG─┬─┘
              └─USING──bind-file─┘             ├─ALL─────┤
                                               └─NO──────┘

►──┬──────────────────────────┬──┬────────────────────────────────────┬──►
   └─COLLECTION──schema-name──┘  └─CALL_RESOLUTION──┬─IMMEDIATE─┬─────┘
                                                    └─DEFERRED──┘

►──┬───────────────────┬──┬─────────────────────┬──┬─────────────────────────────┬──►
   └─CONNECT──┬─1─┬────┘  └─DATETIME──┬─DEF─┬───┘  └─DEFERRED_PREPARE──┬─NO──┬──┘
              └─2─┘                   ├─EUR─┤                          ├─ALL─┤
                                      ├─ISO─┤                          └─YES─┘
                                      ├─JIS─┤
                                      ├─LOC─┤
                                      └─USA─┘

►──┬───────────────────────────────────────┬──┬──────────────────────────────┬──►
   └─DEGREE──┬─1─────────────────────────┬─┘  └─DISCONNECT──┬─EXPLICIT────┬─┘
             ├─degree-of-parallelism─────┤                  ├─AUTOMATIC───┤
             └─ANY───────────────────────┘                  └─CONDITIONAL─┘

►──┬─────────────────────────────────┬──┬──────────────────────┬─────────►
   └─DYNAMICRULES──┬─RUN──────────┬──┘  └─EXPLAIN──┬─NO────┬───┘
                   ├─BIND─────────┤                ├─ALL───┤
                   ├─INVOKERUN────┤                ├─REOPT─┤
                   ├─INVOKEBIND───┤                └─YES───┘
                   ├─DEFINERUN────┤
                   └─DEFINEBIND───┘

►──┬───────────────────────┬──┬─────────────────────────┬──┬─────────────────────────────────┬──►
   └─EXPLSNAP──┬─NO────┬───┘  └─FEDERATED──┬─NO──┬──────┘  │              ┌─,───────────┐    │
               ├─ALL───┤                   └─YES─┘         │              ▼             │    │
               ├─REOPT─┤                                   └─FUNCPATH─────┴─schema-name─┴────┘
               └─YES───┘

►──┬──────────────────┬──┬─────────────────────┬──┬─────────────────────┬──►
   └─GENERIC──string──┘  └─INSERT──┬─DEF─┬─────┘  └─ISOLATION──┬─CS─┬───┘
                                   └─BUF─┘                     ├─RR─┤
                                                               ├─RS─┤
                                                               └─UR─┘
```

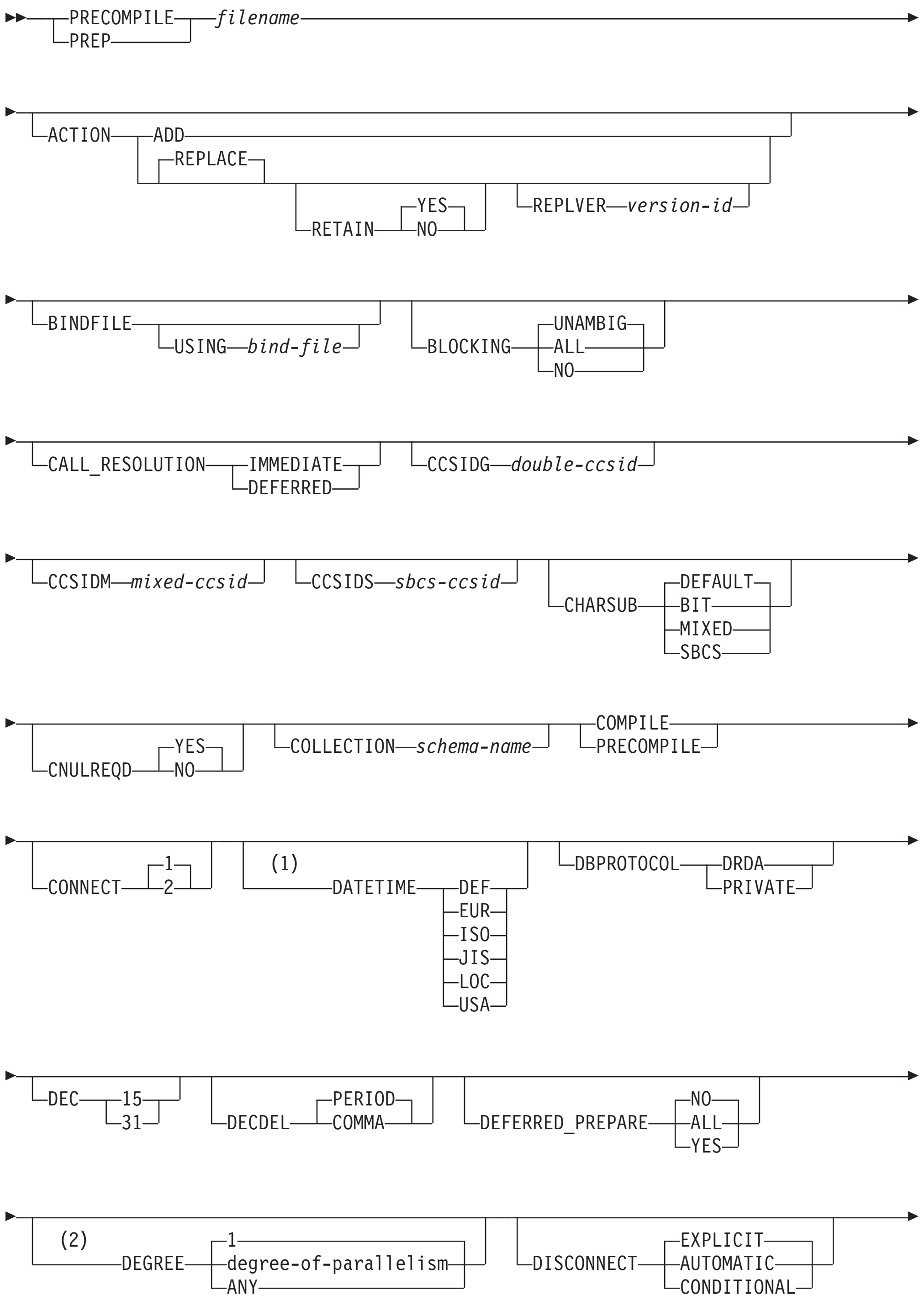
```
►──┬──────────────────────────┬──┬──────────────────────────────┬────────────────►
   └─LANGLEVEL──┬─SAA1───┬─┘  └─LEVEL──consistency token─┘
                ├─MIA────┤
                └─SQL92E─┘

►──┬─────────────────────────────┬──┬──────────────────────────┬──┬─────────────────┬──►
   │ (1)                         │  └─MESSAGES──message-file─┘  └─NOLINEMACRO─┘
   └─────────LONGERROR──┬─NO──┬─┘
                        └─YES─┘

►──┬─────────────────┬──┬──────────────────┬──┬────────────────────────────┬──────►
   └─OPTLEVEL──┬─0─┬─┘  └─OUTPUT──filename─┘  └─OWNER──authorization-id─┘
               └─1─┘

►──┬───────────────────────────────────────┬──────────────────────────────────────►
   └─PACKAGE──┬──────────────────────────┬─┘
              └─USING──package-name─┘

►──┬──────────────────────────────────────────────────┬──┬───────────────────────────────┬──►
   └─PREPROCESSOR──┬─"preprocessor-command"─┬─┘  └─QUALIFIER──qualifier-name─┘
                   └─'preprocessor-command'─┘

                                          ┌─REOPT NONE───┐
►──┬─────────────────────────────────┬──┼──────────────┼──┬────────────────────┬──►
   └─QUERYOPT──optimization-level─┘  ├─REOPT ONCE───┤  └─SQLCA──┬─NONE─┬─┘
                                          └─REOPT ALWAYS─┘           └─SAA──┘

►──┬─────────────────────────────────────┬──┬──────────────────────────────────────┬──►
   │ (2)                                 │  └─SQLFLAG──┬─SQL92E────┬──SYNTAX─┘
   └─────────SQLERROR──┬─NOPACKAGE─┬─┘             ├─MVSDB2V23─┤
                       ├─CHECK─────┤               ├─MVSDB2V31─┤
                       └─CONTINUE──┘               └─MVSDB2V41─┘

►──┬──────────────────────┬──┬──────────────────────┬──┬─────────────────────────────┬──►
   └─SQLRULES──┬─DB2─┬─┘  └─SQLWARN──┬─NO──┬─┘  │                 ┌─NO──┐     │
               └─STD─┘               └─YES─┘     └─STATICREADONLY──┴─YES─┴─┘

►──┬──────────────────────────────┬──┬────────┬──┬──────────────────────────┬──────►
   └─SYNCPOINT──┬─ONEPHASE─┬─┘  └─SYNTAX─┘  └─TARGET──┬─IBMCOB─────┬─┘
                ├─NONE─────┤                            ├─MFCOB──────┤
                └─TWOPHASE─┘                            ├─ANSI_COBOL─┤
                                                        ├─C──────────┤
                                                        ├─CPLUSPLUS──┤
                                                        └─FORTRAN────┘

►──┬──────────────────────────────┬──┬────────────────────────┬───────────────────►
   └─TRANSFORM GROUP──groupname─┘  └─VALIDATE──┬─BIND─┬─┘
                                               └─RUN──┘

►──┬──────────────────────────────┬──┬──────────────────────────────┬────────────►◄
   └─WCHARTYPE──┬─NOCONVERT─┬─┘  └─VERSION──┬─version-id─┬─┘
                └─CONVERT───┘                └─AUTO───────┘
```

Notes:

1 NO is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations for INTEGER columns. YES is the default for 64 bit UNIX systems.

2 SYNTAX is a synonym for SQLERROR(CHECK).

For DB2 on servers other than Windows and UNIX



```
►─┬─────────────────────────────────┬─┬──────────────────────────┬────────────►
  │                ┌─RUN────────┐   │ └─ENCODING──┬─ASCII───┬─┘
  └─DYNAMICRULES───┼─BIND───────┼───┘             ├─EBCDIC──┤
                   ├─INVOKERUN──┤                 ├─UNICODE─┤
                   ├─INVOKEBIND─┤                 └─CCSID───┘
                   ├─DEFINERUN──┤
                   └─DEFINEBIND─┘

►─┬──────────────────────┬─┬──────────────────┬─┬────────────────────────┬────►
  │           ┌─NO──┐    │ └─GENERIC──string──┘ └─IMMEDWRITE───┬─NO──┬──┘
  └─EXPLAIN───┴─YES─┴────┘                                     ├─YES─┤
                                                               └─PH1─┘

►─┬──────────────────────┬─┬─────────────────────────┬─┬─────────────────────────────┬─►
  │               ┌─CS─┐ │ └─KEEPDYNAMIC───┬─YES─┬───┘ └─LEVEL──consistency-token──┘
  └─ISOLATION─────┼─NC─┼─┘                 └─NO──┘
                  ├─RR─┤
                  ├─RS─┤
                  └─UR─┘

►─┬────────────────────────────┬─┬───────────────────────────┬─┬──────────────┬─►
  │   (3)             ┌─NO──┐  │ └─MESSAGES──message-file──┘ └─NOLINEMACRO──┘
  └──────────LONGERROR┴─YES─┴──┘

►─┬─────────────────────┬─┬──────────────────────┬─┬───────────────────────────┬─►
  └─OPTHINT────hint-id──┘ │             ┌─0─┐    │ └─OS400NAMING───┬─SYSTEM─┬──┘
                          └─OPTLEVEL────┴─1─┴────┘                 └─SQL────┘

►─┬──────────────────────────────┬─┬────────────────────────────────────────────┬─►
  └─OWNER──authorization-id──────┘ └─PREPROCESSOR───┬─"preprocessor-command"──┬─┘
                                                    └─'preprocessor-command'──┘

                                                                ┌─REOPT NONE────┐
►─┬──────────────────────────────┬─┬─────────────────────────┬──┼───────────────┼─►
  └─QUALIFIER──qualifier-name────┘ │            ┌─COMMIT─────┐│ ├─REOPT ONCE────┤
                                   └─RELEASE────┴─DEALLOCATE─┴┘ └─REOPT ALWAYS──┘

►─┬─────────────────┬─┬──────────────────────────────────────┬─┬─────────────────────────┬─►
  ├─REOPT VARS──────┤ └─SQLFLAG───┬─SQL92E──────┬──SYNTAX────┘ └─SORTSEQ────┬─JOBRUN─┬──┘
  └─NOREOPT VARS────┘             ├─MVSDB2V23───┤                           └─HEX────┘
                                  ├─MVSDB2V31───┤
                                  └─MVSDB2V41───┘

►─┬──────────────────────┬─┬──────────────────────────────┬─┬─────────────────────────────┬─►
  │            ┌─DB2─┐   │ │              ┌─NOPACKAGE─┐   │ │            ┌─APOSTROPHE─┐   │
  └─SQLRULES───┴─STD─┴───┘ └─SQLERROR─────┼─CHECK─────┼───┘ └─STRDEL─────┴─QUOTE──────┴───┘
                                          └─CONTINUE──┘

►─┬─────────────────────────────┬─┬──────────┬─┬────────────────────────────────┬─►
  │             ┌─ONEPHASE─┐    │ └─SYNTAX───┘ └─TARGET───┬─IBMCOB─────────────┬──┘
  └─SYNCPOINT───┼─NONE─────┼────┘                         ├─MFCOB──────────────┤
                └─TWOPHASE─┘                              ├─ANSI_COBOL─────────┤
                                                          ├─C──────────────────┤
                                                          ├─CPLUSPLUS──────────┤
                                                          ├─FORTRAN────────────┤
                                                          ├─BORLAND_C──────────┤
                                                          └─BORLAND_CPLUSPLUS──┘
```

```
►─┬──────────────┬─┬────────────────────────────┬─┬──────────────────────────┬──────────►
  └─TEXT──label──┘ └─VERSION──┬─version-id─┬────┘ └─VALIDATE──┬─BIND─┬───────┘
                              └─AUTO───────┘                  └─RUN──┘

►─┬───────────────────────────────┬──────────────────────────────────────────────────────►◄
  │               ┌─NOCONVERT─┐   │
  └─WCHARTYPE─────┴─CONVERT───┴───┘
```

Notes:

1 If the server does not support the DATETIME DEF option, it is mapped to DATETIME ISO.

2 The DEGREE option is only supported by DRDA Level 2 Application Servers.

3 NO is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations for INTEGER columns. YES is the default for 64 bit UNIX systems.

Command parameters:

filename

Specifies the source file to be precompiled. An extension of:

- `.sqc` must be specified for C applications (generates a `.c` file)
- `.sqx` (Windows operating systems), or `.sqC` (UNIX based systems) must be specified for C++ applications (generates a `.cxx` file on Windows operating systems, or a `.C` file on UNIX based systems)
- `.sqb` must be specified for COBOL applications (generates a `.cbl` file)
- `.sqf` must be specified for FORTRAN applications (generates a `.for` file on Windows operating systems, or a `.f` file on UNIX based systems).

The preferred extension for C++ applications containing embedded SQL on UNIX based systems is `sqC`; however, the `sqx` convention, which was invented for systems that are not case sensitive, is tolerated by UNIX based systems.

ACTION

Indicates whether the package can be added or replaced.

ADD Indicates that the named package does not exist, and that a new package is to be created. If the package already exists, execution stops, and a diagnostic error message is returned.

REPLACE

Indicates that the existing package is to be replaced by a new one with the same package name and creator. This is the default value for the ACTION option.

RETAIN

Indicates whether EXECUTE authorities are to be preserved when a package is replaced. If ownership of the package changes, the new owner grants the BIND and EXECUTE authority to the previous package owner.

NO Does not preserve EXECUTE authorities when a package is replaced. This value is not supported by DB2.

YES Preserves EXECUTE authorities when a package is replaced. This is the default value.

REPLVER version-id
Replaces a specific version of a package. The version identifier specifies which version of the package is to be replaced. If the specified version does not exist, an error is returned. If the REPLVER option of REPLACE is not specified, and a package already exists that matches the package name and version of the package being precompiled, that package will be replaced; if not, a new package will be added.

BINDFILE
Results in the creation of a bind file. A package is not created unless the **package** option is also specified. If a bind file is requested, but no package is to be created, as in the following example:

```
db2 prep sample.sqc bindfile
```

object existence and authentication SQLCODEs will be treated as warnings instead of errors. This will allow a bind file to be successfully created, even if the database being used for precompilation does not have all of the objects referred to in static SQL statements within the application. The bind file can be successfully bound, creating a package, once the required objects have been created.

USING bind-file
The name of the bind file that is to be generated by the precompiler. The file name must have an extension of `.bnd`. If a file name is not entered, the precompiler uses the name of the program (entered as the *filename* parameter), and adds the `.bnd` extension. If a path is not provided, the bind file is created in the current directory.

BLOCKING
Specifies the type of row blocking for cursors.

ALL Specifies to block for:
- Read-only cursors
- Cursors not specified as FOR UPDATE OF

Ambiguous cursors are treated as read-only.

NO Specifies not to block any cursors. Ambiguous cursors are treated as updatable.

UNAMBIG
Specifies to block for:
- Read-only cursors
- Cursors not specified as FOR UPDATE OF

Ambiguous cursors are treated as updatable.

CALL_RESOLUTION
If set, the CALL_RESOLUTION DEFERRED option indicates that the CALL statement will be executed as an invocation of the deprecated sqleproc() API. If not set or if IMMEDIATE is set, the CALL statement will be executed as a normal SQL statement. SQL0204 will be issued if the precompiler fails to resolve the procedure on a CALL statement with CALL_RESOLUTION IMMEDIATE.

CCSIDG double-ccsid
An integer specifying the coded character set identifier (CCSID) to be used

for double byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDM mixed-ccsid
An integer specifying the coded character set identifier (CCSID) to be used for mixed byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CCSIDS sbcs-ccsid
An integer specifying the coded character set identifier (CCSID) to be used for single byte characters in character column definitions (without a specific CCSID clause) in CREATE and ALTER TABLE SQL statements. This option is not supported by DB2 Database for Linux, UNIX, and Windows. The DRDA server will use a system defined default value if this option is not specified.

CHARSUB
Designates the default character sub-type that is to be used for column definitions in CREATE and ALTER TABLE SQL statements. This DRDA precompile/bind option is not supported by DB2.

BIT Use the FOR BIT DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

DEFAULT
Use the target system defined default in all new character columns for which an explicit sub-type is not specified.

MIXED
Use the FOR MIXED DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

SBCS Use the FOR SBCS DATA SQL character sub-type in all new character columns for which an explicit sub-type is not specified.

CNULREQD
This option is related to the **langlevel** precompile option, which is not supported by DRDA. It is valid only if the bind file is created from a C or a C++ application. This DRDA bind option is not supported by DB2.

NO The application was coded on the basis of the **langlevel** SAA1 precompile option with respect to the null terminator in C string host variables.

YES The application was coded on the basis of the **langlevel** MIA precompile option with respect to the null terminator in C string host variables.

COLLECTION schema-name
Specifies a 30-character collection identifier for the package. If not specified, the authorization identifier for the user processing the package is used.

CONNECT

1 Specifies that a CONNECT statement is to be processed as a type 1 CONNECT.

2 Specifies that a CONNECT statement is to be processed as a type 2 CONNECT.

DATETIME
Specifies the date and time format to be used.

DEF Use a date and time format associated with the territory code of the database.

EUR Use the IBM standard for Europe date and time format.

ISO Use the date and time format of the International Standards Organization.

JIS Use the date and time format of the Japanese Industrial Standard.

LOC Use the date and time format in local form associated with the territory code of the database.

USA Use the IBM standard for U.S. date and time format.

DBPROTOCOL
Specifies what protocol to use when connecting to a remote site that is identified by a three-part name statement. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

DEC Specifies the maximum precision to be used in decimal arithmetic operations. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.

15 15-digit precision is used in decimal arithmetic operations.

31 31-digit precision is used in decimal arithmetic operations.

DECDEL
Designates whether a period (.) or a comma (,) will be used as the decimal point indicator in decimal and floating point literals. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.

COMMA
Use a comma (,) as the decimal point indicator.

PERIOD
Use a period (.) as the decimal point indicator.

DEFERRED_PREPARE
Provides a performance enhancement when accessing DB2 common server databases or DRDA databases. This option combines the SQL PREPARE statement flow with the associated OPEN, DESCRIBE, or EXECUTE statement flow to minimize inter-process or network flow.

NO The PREPARE statement will be executed at the time it is issued.

YES Execution of the PREPARE statement will be deferred until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued.

The PREPARE statement will not be deferred if it uses the INTO clause, which requires an SQLDA to be returned immediately.

However, if the PREPARE INTO statement is issued for a cursor that does not use any parameter markers, the processing will be optimized by pre-OPENing the cursor when the PREPARE is executed.

ALL Same as YES, except that a PREPARE INTO statement is also deferred. If the PREPARE statement uses the INTO clause to return an SQLDA, the application must not reference the content of this SQLDA until the OPEN, DESCRIBE, or EXECUTE statement is issued and returned.

DEGREE
Specifies the degree of parallelism for the execution of static SQL statements in an SMP system. This option does not affect CREATE INDEX parallelism.

1 The execution of the statement will not use parallelism.

degree-of-parallelism
Specifies the degree of parallelism with which the statement can be executed, a value between 2 and 32 767 (inclusive).

ANY Specifies that the execution of the statement can involve parallelism using a degree determined by the database manager.

DISCONNECT

AUTOMATIC
Specifies that all database connections are to be disconnected at commit.

CONDITIONAL
Specifies that the database connections that have been marked RELEASE or have no open WITH HOLD cursors are to be disconnected at commit.

EXPLICIT
Specifies that only database connections that have been explicitly marked for release by the RELEASE statement are to be disconnected at commit.

DYNAMICRULES
Defines which rules apply to dynamic SQL at run time for the initial setting of the values used for authorization ID and for the implicit qualification of unqualified object references.

RUN Specifies that the authorization ID of the user executing the package is to be used for authorization checking of dynamic SQL statements. The authorization ID will also be used as the default package qualifier for implicit qualification of unqualified object references within dynamic SQL statements. This is the default value.

BIND Specifies that all of the rules that apply to static SQL for authorization and qualification are to be used at run time. That is, the authorization ID of the package owner is to be used for authorization checking of dynamic SQL statements, and the default package qualifier is to be used for implicit qualification of unqualified object references within dynamic SQL statements.

DEFINERUN
If the package is used within a routine context, the authorization

ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

DEFINEBIND
If the package is used within a routine context, the authorization ID of the routine definer is to be used for authorization checking and for implicit qualification of unqualified object references within dynamic SQL statements within the routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

INVOKERUN
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES RUN.

INVOKEBIND
If the package is used within a routine context, the current statement authorization ID in effect when the routine is invoked is to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

If the package is used as a standalone application, dynamic SQL statements are processed as if the package were bound with DYNAMICRULES BIND.

Because dynamic SQL statements will be using the authorization ID of the package owner in a package exhibiting bind behavior, the binder of the package should not have any authorities granted to them that the user of the package should not receive. Similarly, when defining a routine that will exhibit define behavior, the definer of the routine should not have any authorities granted to them that the user of the package should not receive since a dynamic statement will be using the authorization ID of the routine's definer.

The following dynamically prepared SQL statements cannot be used within a package that was not bound with DYNAMICRULES RUN: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY, and SET EVENT MONITOR STATE.

ENCODING
Specifies the encoding for all host variables in static statements in the plan or package. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

EXPLAIN
Stores information in the Explain tables about the access plans chosen for each SQL statement in the package. DRDA does not support the ALL value for this option.

NO Explain information will not be captured.

YES Explain tables will be populated with information about the chosen access plan at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA. If this is not done, incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain information for each reoptimizable incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain information will be gathered for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE special register is set to NO.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ALL Explain information for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN MODE special register is set to `NO`.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

EXPLSNAP
Stores Explain Snapshot information in the Explain tables. This DB2 precompile/bind option is not supported by DRDA.

NO An Explain Snapshot will not be captured.

YES An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time for static statements and at run time for incremental bind statements.

If the package is to be used for a routine and the package contains incremental bind statements, then the routine must be defined as MODIFIES SQL DATA or incremental bind statements in the package will cause a run time error (SQLSTATE 42985).

REOPT
Explain Snapshot information for each reoptimizable incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain Snapshot information will be gathered

for reoptimizable dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT special register is set to NO.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, otherwise incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

ALL An Explain Snapshot for each eligible static SQL statement will be placed in the Explain tables at prep/bind time. Explain Snapshot information for each eligible incremental bind SQL statement will be placed in the Explain tables at run time. In addition, Explain Snapshot information will be gathered for eligible dynamic SQL statements at run time, even if the CURRENT EXPLAIN SNAPSHOT special register is set to NO.

If the package is to be used for a routine, then the routine must be defined as MODIFIES SQL DATA, or incremental bind and dynamic statements in the package will cause a run time error (SQLSTATE 42985).

FEDERATED
Specifies whether a static SQL statement in a package references a nickname or a federated view. If this option is not specified and a static SQL statement in the package references a nickname or a federated view, a warning is returned and the package is created.

This option is not supported by DRDA servers.

NO A nickname or federated view is not referenced in the static SQL statements of the package. If a nickname or federated view is encountered in a static SQL statement during the prepare or bind phase of this package, an error is returned and the package is *not* created.

YES A nickname or federated view can be referenced in the static SQL statements of the package. If no nicknames or federated views are encountered in static SQL statements during the prepare or bind of the package, no errors or warnings are returned and the package is created.

FUNCPATH
Specifies the function path to be used in resolving user-defined distinct types and functions in static SQL. If this option is not specified, the default function path is ″SYSIBM″,″SYSFUN″,USER where USER is the value of the USER special register. This DB2 precompile/bind option is not supported by DRDA.

schema-name
An SQL identifier, either ordinary or delimited, which identifies a schema that exists at the application server. No validation that the schema exists is made at precompile or at bind time. The same schema cannot appear more than once in the function path. The number of schemas that can be specified is limited by the length of the resulting function path, which cannot exceed 254 bytes. The schema SYSIBM does not need to be explicitly specified; it is implicitly assumed to be the first schema if it is not included in the function path.

INSERT
Allows a program being precompiled or bound against a DB2 Enterprise Server Edition server to request that data inserts be buffered to increase performance.

BUF Specifies that inserts from an application should be buffered.

DEF Specifies that inserts from an application should not be buffered.

GENERIC string
Supports the binding of new options that are defined in the target database, but are not supported by DRDA. Do not use this option to pass bind options that *are* defined in BIND or PRECOMPILE. This option can substantially improve dynamic SQL performance. The syntax is as follows:

```
   generic "option1 value1 option2 value2 ..."
```

Each option and value must be separated by one or more blank spaces. For example, if the target DRDA database is DB2 Universal Database, Version 8, one could use:

```
   generic "explsnap all queryopt 3 federated yes"
```

to bind each of the EXPLSNAP, QUERYOPT, and FEDERATED options.

The maximum length of the string is 1023 bytes.

IMMEDWRITE
Indicates whether immediate writes will be done for updates made to group buffer pool dependent pagesets or database partitions. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

ISOLATION
Determines how far a program bound to this package can be isolated from the effect of other executing programs.

CS Specifies Cursor Stability as the isolation level.

NC No Commit. Specifies that commitment control is not to be used. This isolation level is not supported by DB2.

RR Specifies Repeatable Read as the isolation level.

RS Specifies Read Stability as the isolation level. Read Stability ensures that the execution of SQL statements in the package is isolated from other application processes for rows read and changed by the application.

UR Specifies Uncommitted Read as the isolation level.

LANGLEVEL
Specifies the SQL rules that apply for both the syntax and the semantics for both static and dynamic SQL in the application. This option is not supported by DRDA servers.

MIA Select the ISO/ANS SQL92 rules as follows:

- To support error SQLCODE or SQLSTATE checking, an SQLCA must be declared in the application code.
- C null-terminated strings are padded with blanks and always include a null-terminating character, even if truncation occurs.
- The FOR UPDATE clause is optional for all columns to be updated in a positioned UPDATE.

- A searched UPDATE or DELETE requires SELECT privilege on the object table of the UPDATE or DELETE statement if a column of the object table is referenced in the search condition or on the right hand side of the assignment clause.
- A column function that can be resolved using an index (for example MIN or MAX) will also check for nulls and return warning SQLSTATE 01003 if there were any nulls.
- An error is returned when a duplicate unique constraint is included in a CREATE or ALTER TABLE statement.
- An error is returned when no privilege is granted and the grantor has no privileges on the object (otherwise a warning is returned).

SAA1 Select the common IBM DB2 rules as follows:

- To support error SQLCODE or SQLSTATE checking, an SQLCA must be declared in the application code.
- C null-terminated strings are not terminated with a null character if truncation occurs.
- The FOR UPDATE clause is required for all columns to be updated in a positioned UPDATE.
- A searched UPDATE or DELETE will not require SELECT privilege on the object table of the UPDATE or DELETE statement unless a fullselect in the statement references the object table.
- A column function that can be resolved using an index (for example MIN or MAX) will not check for nulls and warning SQLSTATE 01003 is not returned.
- A warning is returned and the duplicate unique constraint is ignored.
- An error is returned when no privilege is granted.

SQL92E
Defines the ISO/ANS SQL92 rules as follows:

- To support checking of SQLCODE or SQLSTATE values, variables by this name can be declared in the host variable declare section (if neither is declared, SQLCODE is assumed during precompilation).
- C null-terminated strings are padded with blanks and always include a null-terminating character, even if truncation occurs.
- The FOR UPDATE clause is optional for all columns to be updated in a positioned UPDATE.
- A searched UPDATE or DELETE requires SELECT privilege on the object table of the UPDATE or DELETE statement if a column of the object table is referenced in the search condition or on the right hand side of the assignment clause.
- A column function that can be resolved using an index (for example MIN or MAX) will also check for nulls and return warning SQLSTATE 01003 if there were any nulls.
- An error is returned when a duplicate unique constraint is included in a CREATE or ALTER TABLE statement.
- An error is returned when no privilege is granted and the grantor has no privileges on the object (otherwise a warning is returned).

KEEPDYNAMIC
Specifies whether dynamic SQL statements are to be kept after commit points. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

LEVEL consistency-token
Defines the level of a module using the consistency token. The consistency token is any alphanumeric value up to 8 characters in length. The RDB package consistency token verifies that the requester's application and the relational database package are synchronized. This option is not recommended for general use.

LONGERROR
Indicates whether long host variable declarations will be treated as an error. For portability, sqlint32 can be used as a declaration for an INTEGER column in precompiled C and C++ code.

NO Does not generate errors for the use of long host variable declarations. This is the default for 32 bit systems and for 64 bit NT systems where long host variables can be used as declarations for INTEGER columns. The use of this option on 64 bit UNIX platforms will allow long host variables to be used as declarations for BIGINT columns.

YES Generates errors for the use of long host variable declarations. This is the default for 64 bit UNIX systems.

MESSAGES message-file
Specifies the destination for warning, error, and completion status messages. A message file is created whether the bind is successful or not. If a message file name is not specified, the messages are written to standard output. If the complete path to the file is not specified, the current directory is used. If the name of an existing file is specified, the contents of the file are overwritten.

NOLINEMACRO
Suppresses the generation of the #line macros in the output `.c` file. Useful when the file is used with development tools which require source line information such as profiles, cross-reference utilities, and debuggers. This precompile option is used for the C/C^{++} programming languages only.

OPTHINT
Controls whether query optimization hints are used for static SQL. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

OPTLEVEL
Indicates whether the C/C++ precompiler is to optimize initialization of internal SQLDAs when host variables are used in SQL statements. Such optimization can increase performance when a single SQL statement (such as FETCH) is used inside a tight loop.

0 Instructs the precompiler not to optimize SQLDA initialization.

1 Instructs the precompiler to optimize SQLDA initialization. This value should not be specified if the application uses:

- pointer host variables, as in the following example:

```
exec sql begin declare section;
char (*name)[20];
short *id;
exec sql end declare section;
```

- C++ data members directly in SQL statements.

OUTPUT filename
Overrides the default name of the modified source file produced by the compiler. It can include a path.

OS400NAMING
Specifies which naming option is to be used when accessing DB2 UDB for iSeries data. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

Because of the slashes used as separators, a DB2 utility can still report a syntax error at execution time on certain SQL statements which use the iSeries system naming convention, even though the utility might have been precompiled or bound with the OS400NAMING SYSTEM option. For example, the Command Line Processor will report a syntax error on an SQL CALL statement if the iSeries system naming convention is used, whether or not it has been precompiled or bound using the OS400NAMING SYSTEM option.

OWNER authorization-id
Designates a 30-character authorization identifier for the package owner. The owner must have the privileges required to execute the SQL statements contained in the package. Only a user with SYSADM or DBADM authority can specify an authorization identifier other than the user ID. The default value is the primary authorization ID of the precompile/bind process. SYSIBM, SYSCAT, and SYSSTAT are not valid values for this option.

PACKAGE
Creates a package. If neither **package**, **bindfile**, nor **syntax** is specified, a package is created in the database by default.

USING package-name
The name of the package that is to be generated by the precompiler. If a name is not entered, the name of the application program source file (minus extension and folded to uppercase) is used. Maximum length is 8 characters.

PREPROCESSOR ″preprocessor-command″

Specifies the preprocessor command that can be executed by the precompiler before it processes embedded SQL statements. The preprocessor command string (maximum length 1024 bytes) must be enclosed either by double or by single quotation marks.

This option enables the use of macros within the declare section. A valid preprocessor command is one that can be issued from the command line to invoke the preprocessor without specifying a source file. For example,

```
xlc -P -DMYMACRO=0
```

QUALIFIER qualifier-name
Provides an 30-character implicit qualifier for unqualified objects contained in the package. The default is the owner’s authorization ID, whether or not **owner** is explicitly specified.

QUERYOPT optimization-level
Indicates the desired level of optimization for all static SQL statements contained in the package. The default value is 5. The SET CURRENT

QUERY OPTIMIZATION statement describes the complete range of optimization levels available. This DB2 precompile/bind option is not supported by DRDA.

RELEASE
Indicates whether resources are released at each COMMIT point, or when the application terminates. This DRDA precompile/bind option is not supported by DB2.

COMMIT
Release resources at each COMMIT point. Used for dynamic SQL statements.

DEALLOCATE
Release resources only when the application terminates.

REOPT
Specifies whether to have DB2 optimize an access path using values for host variables, parameter markers, and special registers. Valid values are:

NONE
The access path for a given SQL statement containing host variables, parameter markers or special registers will not be optimized using real values for these variables. The default estimates for the these variables will be used instead, and this plan is cached and used subsequently. This is the default behavior.

ONCE The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers or special registers when the query is first executed. This plan is cached and used subsequently.

ALWAYS
The access path for a given SQL statement will always be compiled and reoptimized using the values of the host variables, parameter markers or special registers known at each execution time.

REOPT / NOREOPT VARS
These options have been replaced by REOPT ALWAYS and REOPT NONE; however, they are still supported for compatibility with previous releases. Specifies whether to have DB2 determine an access path at run time using values for host variables, parameter markers, and special registers. Supported by DB2 for OS/390 only. For a list of supported option values, refer to the documentation for DB2 for OS/390.

SQLCA
For FORTRAN applications only. This option is ignored if it is used with other languages.

NONE
Specifies that the modified source code is not consistent with the SAA definition.

SAA Specifies that the modified source code is consistent with the SAA definition.

SQLERROR
Indicates whether to create a package or a bind file if an error is encountered.

CHECK
Specifies that the target system performs all syntax and semantic

checks on the SQL statements being bound. A package will not be created as part of this process. If, while binding, an existing package with the same name and version is encountered, the existing package is neither dropped nor replaced even if **action replace** was specified.

CONTINUE
Creates a package, even if errors occur when binding SQL statements. Those statements that failed to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to execute them at run time generates an error (SQLCODE -525, SQLSTATE 51015).

NOPACKAGE
A package or a bind file is not created if an error is encountered.

SQLFLAG
Identifies and reports on deviations from the SQL language syntax specified in this option.

A bind file or a package is created only if the **bindfile** or the **package** option is specified, in addition to the **sqlflag** option.

Local syntax checking is performed only if one of the following options is specified:
- **bindfile**
- **package**
- **sqlerror check**
- **syntax**

If **sqlflag** is not specified, the flagger function is not invoked, and the bind file or the package is not affected.

SQL92E SYNTAX
The SQL statements will be checked against ANSI or ISO SQL92 Entry level SQL language format and syntax with the exception of syntax rules that would require access to the database catalog. Any deviation is reported in the precompiler listing.

MVSDB2V23 SYNTAX
The SQL statements will be checked against MVS DB2 Version 2.3 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.

MVSDB2V31 SYNTAX
The SQL statements will be checked against MVS DB2 Version 3.1 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.

MVSDB2V41 SYNTAX
The SQL statements will be checked against MVS DB2 Version 4.1 SQL language syntax. Any deviation from the syntax is reported in the precompiler listing.

SORTSEQ
Specifies which sort sequence table to use on the iSeries system. Supported by DB2 UDB for iSeries only. For a list of supported option values, refer to the documentation for DB2 for iSeries.

SQLRULES
Specifies:

- Whether type 2 CONNECTs are to be processed according to the DB2 rules or the Standard (STD) rules based on ISO/ANS SQL92.
- How a user or application can specify the format of LOB answer set columns.

DB2

- Permits the SQL CONNECT statement to switch the current connection to another established (*dormant*) connection.
- The user or application can specify the format of a LOB column only during the first fetch request.

STD

- Permits the SQL CONNECT statement to establish a *new* connection only. The SQL SET CONNECTION statement must be used to switch to a dormant connection.
- The user or application can change the format of a LOB column with each fetch request.

SQLWARN
Indicates whether warnings will be returned from the compilation of dynamic SQL statements (via PREPARE or EXECUTE IMMEDIATE), or from describe processing (via PREPARE...INTO or DESCRIBE).

NO Warnings will not be returned from the SQL compiler.

YES Warnings will be returned from the SQL compiler.

SQLCODE +238 is an exception. It is returned regardless of the **sqlwarn** option value.

STATICREADONLY
Determines whether static cursors will be treated as being READ ONLY. This DB2 precompile/bind option is not supported by DRDA.

NO All static cursors will take on the attributes as would normally be generated given the statement text and the setting of the LANGLEVEL precompile option. This is the default value.

YES Any static cursor that does not contain the FOR UPDATE or FOR READ ONLY clause will be considered READ ONLY.

STRDEL
Designates whether an apostrophe (′) or double quotation marks (") will be used as the string delimiter within SQL statements. This DRDA precompile/bind option is not supported by DB2. The DRDA server will use a system defined default value if this option is not specified.

APOSTROPHE
Use an apostrophe (′) as the string delimiter.

QUOTE
Use double quotation marks (") as the string delimiter.

SYNCPOINT
Specifies how commits or rollbacks are to be coordinated among multiple database connections. This command parameter is ignored and is only included here for backward compatibility.

NONE
: Specifies that no Transaction Manager (TM) is to be used to perform a two-phase commit, and does not enforce single updater, multiple reader. A COMMIT is sent to each participating database. The application is responsible for recovery if any of the commits fail.

ONEPHASE
: Specifies that no TM is to be used to perform a two-phase commit. A one-phase commit is to be used to commit the work done by each database in multiple database transactions.

TWOPHASE
: Specifies that the TM is required to coordinate two-phase commits among those databases that support this protocol.

SYNTAX

Suppresses the creation of a package or a bind file during precompilation. This option can be used to check the validity of the source file without modifying or altering existing packages or bind files. **Syntax** is a synonym for **sqlerror check**.

If **syntax** is used together with the **package** option, **package** is ignored.

TARGET

Instructs the precompiler to produce modified code tailored to one of the supported compilers on the current platform.

IBMCOB
: On AIX, code is generated for the IBM COBOL Set for AIX compiler.

MFCOB
: Code is generated for the Micro Focus COBOL compiler. This is the default if a **target** value is not specified with the COBOL precompiler on all UNIX operating systems and Windows.

ANSI_COBOL
: Code compatible with the ANS X3.23-1985 standard is generated.

C
: Code compatible with the C compilers supported by DB2 on the current platform is generated.

CPLUSPLUS
: Code compatible with the C++ compilers supported by DB2 on the current platform is generated.

FORTRAN
: Code compatible with the FORTRAN compilers supported by DB2 on the current platform is generated.

TEXT label

The description of a package. Maximum length is 255 characters. The default value is blanks. This DRDA precompile/bind option is not supported by DB2.

TRANSFORM GROUP

Specifies the transform group name to be used by static SQL statements for exchanging user-defined structured type values with host programs. This transform group is not used for dynamic SQL statements or for the exchange of parameters and results with external functions or methods. This option is not supported by DRDA servers.

groupname

An SQL identifier of up to 18 characters in length. A group name cannot include a qualifier prefix and cannot begin with the prefix SYS since this is reserved for database use. In a static SQL statement that interacts with host variables, the name of the transform group to be used for exchanging values of a structured type is as follows:

- The group name in the TRANSFORM GROUP bind option, if any
- The group name in the TRANSFORM GROUP prep option as specified at the original precompilation time, if any
- The DB2_PROGRAM group, if a transform exists for the given type whose group name is DB2_PROGRAM
- No transform group is used if none of the above conditions exist.

The following errors are possible during the bind of a static SQL statement:

- SQLCODE yyy, SQLSTATE xxxxx: A transform is needed, but no static transform group has been selected.
- SQLCODE yyy, SQLSTATE xxxxx: The selected transform group does not include a necessary transform (TO SQL for input variables, FROM SQL for output variables) for the data type that needs to be exchanged.
- SQLCODE yyy, SQLSTATE xxxxx: The result type of the FROM SQL transform is not compatible with the type of the output variable, or the parameter type of the TO SQL transform is not compatible with the type of the input variable.

In these error messages, *yyyyy* is replaced by the SQL error code, and *xxxxx* by the SQL state code.

VALIDATE

Determines when the database manager checks for authorization errors and object not found errors. The package owner authorization ID is used for validity checking.

BIND Validation is performed at precompile/bind time. If all objects do not exist, or all authority is not held, error messages are produced. If **sqlerror continue** is specified, a package/bind file is produced despite the error message, but the statements in error are not executable.

RUN Validation is attempted at bind time. If all objects exist, and all authority is held, no further checking is performed at execution time.

If all objects do not exist, or all authority is not held at precompile/bind time, warning messages are produced, and the package is successfully bound, regardless of the **sqlerror continue** option setting. However, authority checking and existence checking for SQL statements that failed these checks during the precompile/bind process can be redone at execution time.

VERSION

Defines the version identifier for a package. If this option is not specified, the package version will be "" (the empty string).

version-id
Specifies a version identifier that is any alphanumeric value, $, #, @, _, -, or ., up to 64 characters in length.

AUTO
The version identifier will be generated from the consistency token. If the consistency token is a timestamp (it will be if the LEVEL option is not specified), the timestamp is converted into ISO character format and is used as the version identifier.

WCHARTYPE
Specifies the format for graphic data.

CONVERT
Host variables declared using the wchar_t base type will be treated as containing data in wchar_t format. Since this format is not directly compatible with the format of graphic data stored in the database (DBCS format), input data in wchar_t host variables is implicitly converted to DBCS format on behalf of the application, using the ANSI C function `wcstombs()`. Similarly, output DBCS data is implicitly converted to wchar_t format, using `mbstowcs()`, before being stored in host variables.

NOCONVERT
Host variables declared using the wchar_t base type will be treated as containing data in DBCS format. This is the format used within the database for graphic data; it is, however, different from the native wchar_t format implemented in the C language. Using NOCONVERT means that graphic data will not undergo conversion between the application and the database, which can improve efficiency. The application is, however, responsible for ensuring that data in wchar_t format is not passed to the database manager. When this option is used, wchar_t host variables should not be manipulated with the C wide character string functions, and should not be initialized with wide character literals (*L-literals*).

Usage notes:

A modified source file is produced, which contains host language equivalents to the SQL statements. By default, a package is created in the database to which a connection has been established. The name of the package is the same as the file name (minus the extension and folded to uppercase), up to a maximum of 8 characters.

Following connection to a database, PREP executes under the transaction that was started. PREP then issues a COMMIT or a ROLLBACK to terminate the current transaction and start another one.

Creating a package with a schema name that does not already exist results in the implicit creation of that schema. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

During precompilation, an Explain Snapshot is not taken unless a package is created and **explsnap** has been specified. The snapshot is put into the Explain tables of the user creating the package. Similarly, Explain table information is only captured when **explain** is specified, and a package is created.

Precompiling stops if a fatal error or more than 100 errors occur. If a fatal error occurs, the utility stops precompiling, attempts to close all files, and discards the package.

When a package exhibits bind behavior, the following will be true:

1. The implicit or explicit value of the BIND option OWNER will be used for authorization checking of dynamic SQL statements.
2. The implicit or explicit value of the BIND option QUALIFIER will be used as the implicit qualifier for qualification of unqualified objects within dynamic SQL statements.
3. The value of the special register CURRENT SCHEMA has no effect on qualification.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior as specified by the DYNAMICRULES option for that specific package and the environment they are used in.

If an SQL statement was found to be in error and the PRECOMPILE option SQLERROR CONTINUE was specified, the statement will be marked as invalid and another PRECOMPILE must be issued in order to change the state of the SQL statement. Implicit and explicit rebind will not change the state of an invalid statement in a package bound with VALIDATE RUN. A statement can change from static to incremental bind or incremental bind to static across implicit and explicit rebinds depending on whether or not object existence or authority problems exist during the rebind.

Binding a package with REOPT ONCE or REOPT ALWAYS might change static and dynamic statement compilation and performance.

Related concepts:

- "Authorization considerations for dynamic SQL" on page 1307
- "Effect of DYNAMICRULES bind option on dynamic SQL" on page 1308
- "Performance improvements when using REOPT option of the BIND command" in *Developing Embedded SQL Applications*
- "WCHARTYPE precompiler option for graphic data in C and C++ embedded SQL applications" in *Developing Embedded SQL Applications*

Related tasks:

- "Specifying row blocking to reduce overhead" in *Performance Guide*

Related reference:

- "BIND" on page 441
- "Datetime values" in *SQL Reference, Volume 1*
- "SET CURRENT QUERY OPTIMIZATION statement" in *SQL Reference, Volume 2*
- "sqlaprep - Precompile application program" on page 831

REBIND

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization:

One of the following:
- *sysadm* or *dbadm* authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

The authorization ID logged in the BOUNDBY column of the `SYSCAT.PACKAGES` system catalog table, which is the ID of the most recent binder of the package, is used as the binder authorization ID for the rebind, and for the default *schema* for table references in the package. This default qualifier can be different from the authorization ID of the user executing the rebind request. REBIND will use the same bind options that were specified when the package was created.

Required connection:

Database. If no database connection exists, and if implicit connect is enabled, a connection to the default database is made.

Command syntax:

```
►►──REBIND──┬─────────┬──package-name──┬──────────────────────┬──────────────►
            └─PACKAGE─┘                └─VERSION──version-name─┘

►─RESOLVE──┬─ANY──────────┬──┬──────────────┬──────────────────────────────►◄
           └─CONSERVATIVE─┘  ├─REOPT NONE───┤
                             ├─REOPT ONCE───┤
                             └─REOPT ALWAYS─┘
```

Command parameters:

PACKAGE package-name
: The qualified or unqualified name that designates the package to be rebound.

VERSION version-name
: The specific version of the package to be rebound. When the version is not specified, it is taken to be "" (the empty string).

RESOLVE
: Specifies whether rebinding of the package is to be performed with or without conservative binding semantics. This affects whether new functions and data types are considered during function resolution and type resolution on static DML statements in the package. This option is not supported by DRDA. Valid values are:

ANY Any of the functions and types in the SQL path are considered for function and type resolution. Conservative binding semantics are not used. This is the default.

CONSERVATIVE
Only functions and types in the SQL path that were defined before

the last explicit bind time stamp are considered for function and type resolution. Conservative binding semantics are used. This option is not supported for an inoperative package.

REOPT
Specifies whether to have DB2 optimize an access path using values for host variables, parameter markers, and special registers.

NONE
The access path for a given SQL statement containing host variables, parameter markers or special registers will not be optimized using real values for these variables. The default estimates for the these variables will be used instead, and this plan is cached and used subsequently. This is the default behavior.

ONCE The access path for a given SQL statement will be optimized using the real values of the host variables, parameter markers or special registers when the query is first executed. This plan is cached and used subsequently.

ALWAYS
The access path for a given SQL statement will always be compiled and reoptimized using the values of the host variables, parameter markers or special registers known at each execution time.

Usage notes:

REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables ″what if″ analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. It also permits multiple rebinds within a unit of work.

The REBIND command *will* commit the transaction if auto-commit is enabled.

This command:

- Provides a quick way to recreate a package. This enables the user to take advantage of a change in the system without a need for the original bind file. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, the REBIND command can be used to recreate the package. REBIND can also be used to recreate packages after RUNSTATS has been executed, thereby taking advantage of the new statistics.
- Provides a method to recreate inoperative packages. Inoperative packages must be explicitly rebound by invoking either the bind utility or the rebind utility. A package will be marked inoperative (the VALID column of the `SYSCAT.PACKAGES` system catalog will be set to `X`) if a function instance on which the package depends is dropped.
- Gives users control over the rebinding of invalid packages. Invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This might result in a noticeable delay in the execution of the first SQL request for the invalid package. It may be desirable to explicitly rebind invalid packages, rather than allow the system to automatically rebind them, in order to eliminate the initial delay and to prevent unexpected SQL error messages which might be returned in case the implicit rebind fails. For example, following migration, all packages stored in the database will be invalidated by the DB2 Version 8 migration process. Given that this might involve a large number of

packages, it may be desirable to explicitly rebind all of the invalid packages at one time. This explicit rebinding can be accomplished using BIND, REBIND, or the **db2rbind** tool).

If multiple versions of a package (many versions with the same package name and creator) exist, only one version can be rebound at once. If not specified in the VERSION option, the package version defaults to be "". Even if there exists only one package with a name that matches, it will not be rebound unless its version matches the one specified or the default.

The choice of whether to use BIND or REBIND to explicitly rebind a package depends on the circumstances. It is recommended that REBIND be used whenever the situation does not specifically require the use of BIND, since the performance of REBIND is significantly better than that of BIND. BIND *must* be used, however:

- When there have been modifications to the program (for example, when SQL statements have been added or deleted, or when the package does not match the executable for the program).
- When the user wishes to modify any of the bind options as part of the rebind. REBIND does not support any bind options. For example, if the user wishes to have privileges on the package granted as part of the bind process, BIND must be used, since it has a **grant** option.
- When the package does not currently exist in the database.
- When detection of *all* bind errors is desired. REBIND only returns the first error it detects, whereas the BIND command returns the first 100 errors that occur during binding.

REBIND is supported by DB2 Connect.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the `SYSCAT.PACKAGES` system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the `SYSCAT.STATEMENTS` system catalog table.

If REBIND encounters an error, processing stops, and an error message is returned.

REBIND will re-explain packages that were created with the **explsnap** bind option set to `YES` or `ALL` (indicated in the EXPLAIN_SNAPSHOT column in the `SYSCAT.PACKAGES` catalog table entry for the package) or with the **explain** bind option set to `YES` or `ALL` (indicated in the EXPLAIN_MODE column in the `SYSCAT.PACKAGES` catalog table entry for the package). The Explain tables used are those of the REBIND requester, not the original binder.

If an SQL statement was found to be in error and the BIND option SQLERROR CONTINUE was specified, the statement will be marked as invalid even if the problem has been corrected. REBIND will not change the state of an invalid statement. In a package bound with VALIDATE RUN, a statement can change from static to incremental bind or incremental bind to static across a REBIND depending on whether or not object existence or authority problems exist during the REBIND.

Rebinding a package with REOPT ONCE/ALWAYS might change static and dynamic statement compilation and performance.

If REOPT is not specified, REBIND will preserve the existing REOPT value used at precompile or bind time.

Related reference:

- "BIND" on page 441
- "RUNSTATS command" in *Command Reference*
- "db2rbind - Rebind all packages " on page 489
- "sqlarbnd - Rebind package" on page 833

Part 10. Application programming interfaces (APIs)

Following are the application programming interfaces (APIs) that correspond to the DB2 UDB commands that are used for the Common Criteria evaluation. Note that:

- APIs are not used for the Common Criteria certification. The APIs are included in this document for reasons of completeness only.
- Not every API has a corresponding command, and vice versa.

Chapter 38. DB2 UDB APIs for Administrators

db2Backup - Back up a database or table space

Creates a backup copy of a database or a table space.

Scope:

This API only affects the database partition on which it is executed.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint

Required connection:

Database. This API automatically establishes a connection to the specified database.

The connection will be terminated upon the completion of the backup.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Backup (
       db2Uint32 versionNumber,
       void * pDB2BackupStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2BackupStruct
{
  char *piDBAlias;
  char oApplicationId[SQLU_APPLID_LEN+1];
  char oTimestamp[SQLU_TIME_STAMP_LEN+1];
  struct db2TablespaceStruct *piTablespaceList;
  struct db2MediaListStruct *piMediaList;
  char *piUsername;
  char *piPassword;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 oBackupSize;
  db2Uint32 iCallerAction;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iParallelism;
  db2Uint32 iOptions;
  db2Uint32 iUtilImpactPriority;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32 numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32 numLocations;
  char locationType;
} db2MediaListStruct;

SQL_API_RC SQL_API_FN
  db2gBackup (
       db2Uint32 versionNumber,
       void * pDB2gBackupStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gBackupStruct
{
  char *piDBAlias;
  db2Uint32 iDBAliasLen;
  char *poApplicationId;
  db2Uint32 iApplicationIdLen;
  char *poTimestamp;
  db2Uint32 iTimestampLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct *piMediaList;
  char *piUsername;
  db2Uint32 iUsernameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 oBackupSize;
  db2Uint32 iCallerAction;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iParallelism;
  db2Uint32 iOptions;
  db2Uint32 iUtilImpactPriority;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char *tablespaces;
  db2Uint32 numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char *locations;
  db2Uint32 numLocations;
  char locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char *pioData;
   db2Uint32 iLength;
   db2Uint32 oLength;
} db2Char;
```

db2Backup API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2BackupStruct.

pDB2BackupStruct
Input. A pointer to the db2BackupStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2BackupStruct data structure parameters:

piDBAlias
Input. A string containing the database alias (as cataloged in the system database directory) of the database to back up.

oApplicationId
Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

oTimestamp
Output. The API will return the time stamp of the backup image

piTablespaceList
Input. List of table spaces to be backed up. Required for table space level backup only. Must be NULL for a database level backup. See structure db2TablespaceStruct.

piMediaList
: Input. This structure allows the caller to specify the destination for the backup operation. The information provided depends on the value of the locationType parameter. The valid values for locationType parameter (defined in sqlutil header file, located in the include directory) are:

  SQLU_LOCAL_MEDIA
  : Local devices (a combination of tapes, disks, or diskettes).

  SQLU_TSM_MEDIA
  : TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.

  SQLU_OTHER_MEDIA
  : Vendor product. Provide the shared library name in the locations field.

  SQLU_USER_EXIT
  : User exit. No additional input is required (only available when server is on OS/2).

  For more information, see the db2MediaListStruct structure.

piUsername
: Input. A string containing the user name to be used when attempting a connection. Can be NULL.

piPassword
: Input. A string containing the password to be used with the user name. Can be NULL.

piVendorOptions
: Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and code page is not checked for this data.

iVendorOptionsSize
: Input. The length of the piVendorOptions field, which cannot exceed 65535 bytes.

oBackupSize
: Output. Size of the backup image (in MB).

iCallerAction
: Input. Specifies action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

  DB2BACKUP_BACKUP
  : Start the backup.

  DB2BACKUP_NOINTERRUPT
  : Start the backup. Specifies that the backup will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the backup have been mounted, and utility prompts are not desired.

DB2BACKUP_CONTINUE
Continue the backup after the user has performed some action requested by the utility (mount a new tape, for example).

DB2BACKUP_TERMINATE
Terminate the backup after the user has failed to perform some action requested by the utility.

DB2BACKUP_DEVICE_TERMINATE
Remove a particular device from the list of devices used by backup. When a particular medium is full, backup will return a warning to the caller (while continuing to process using the remaining devices). Call backup again with this caller action to remove the device which generated the warning from the list of devices being used.

DB2BACKUP_PARM_CHK
Used to validate parameters without performing a backup. This option does not terminate the database connection after the call returns. After successful return of this call, it is expected that the user will issue a call with SQLUB_CONTINUE to proceed with the action.

DB2BACKUP_PARM_CHK_ONLY
Used to validate parameters without performing a backup. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

iBufferSize
Input. Backup buffer size in 4 KB allocation units (pages). Minimum is 8 units.

iNumBuffers
Input. Specifies number of backup buffers to be used. Minimum is 2. Maximum is limited by memory.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024.

iOptions
Input. A bitmap of backup properties. The options are to be combined using the bitwise OR operator to produce a value for iOptions. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2BACKUP_OFFLINE
Offline gives an exclusive connection to the database.

DB2BACKUP_ONLINE
Online allows database access by other applications while the backup operation occurs.

Note: An online backup operation may appear to hang if users are holding locks on SMS LOB data.

DB2BACKUP_DB
Full database backup.

DB2BACKUP_TABLESPACE
Table space level backup. For a table space level backup, provide a list of table spaces in the piTablespaceList parameter.

DB2BACKUP_INCREMENTAL
Specifies a cumulative (incremental) backup image. An incremental backup image is a copy of all database data that has changed since the most recent successful, full backup operation.

DB2BACKUP_DELTA
Specifies a noncumulative (delta) backup image. A delta backup image is a copy of all database data that has changed since the most recent successful backup operation of any type.

DB2BACKUP_COMPRESS
Specifies that the backup should be compressed.

DB2BACKUP_INCLUDE_COMPR_LIB
Specifies that the library used for compressing the backup should be included in the backup image.

DB2BACKUP_EXCLUDE_COMPR_LIB
Specifies that the library used for compressing the backup should be not included in the backup image.

DB2BACKUP_INCLUDE_LOGS
Specifies that the backup image should also include the range of log files required to restore and roll forward this image to some consistent point in time. This option is not valid for an offline backup or a multi-partition backup.

DB2BACKUP_EXCLUDE_LOGS
Specifies that the backup image should not include any log files.

Note: When performing an offline backup operation, logs are excluded whether or not this option is specified.

iUtilImpactPriority
Input. Specifies the priority value to be used during a backup.

- If this value is non-zero, the utility will run throttled. Otherwise, the utility will run unthrottled.
- If there are multiple concurrent utilities running, this parameter is used to determine a relative priority between the throttled tasks. For example, consider two concurrent backups, one with priority 2 and another with priority 4. Both will be throttled, but the one with priority 4 will be allotted more resources. Setting priorities to 2 and 4 is no different than setting them to 5 and 10 or 30 and 60. Priorities values are purely relative.

piComprLibrary
Input. Indicates the name of the external library to be used to perform compression of the backup image. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will use the default library for compression. If the specified library is not found, the backup will fail.

piComprOptions
Input. Describes a block of binary data that will be passed to the initialization routine in the compression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with

the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize
Input. A four-byte unsigned integer representing the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

db2TablespaceStruct data structure specific parameters:

tablespaces
Input. A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numTablespaces
Input. Number of entries in the tablespaces parameter.

db2MediaListStruct data structure parameters:

locations
Input. A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
Input. The number of entries in the locations parameter.

locationType
Input. A character indicating the media type. Valid values (defined in sqlutil header file, located in the include directory.) are:

SQLU_LOCAL_MEDIA
Local devices (tapes, disks, diskettes, or named pipes).

SQLI_XBSA_MEDIA
XBSA interface.

SQLU_TSM_MEDIA
Tivoli Storage Manager.

SQLU_OTHER_MEDIA
Vendor library.

SQLU_USER_EXIT
User exit (only available when the server is on OS/2).

db2gBackupStruct data structure specific parameters:

iDBAliasLen
Input. A 4-byte unsigned integer representing the length in bytes of the database alias.

iApplicationIdLen
Input. A 4-byte unsigned integer representing the length in bytes of the poApplicationId buffer. Should be equal to SQLU_APPLID_LEN+1 (defined in sqlutil.h).

iTimestampLen
Input. A 4-byte unsigned integer representing the length in bytes of the poTimestamp buffer. Should be equal to SQLU_TIME_STAMP_LEN+1 (defined in sqlutil.h).

iUsernameLen
: Input. A 4-byte unsigned integer representing the length in bytes of the user name. Set to zero if no user name is provided.

iPasswordLen
: Input. A 4-byte unsigned integer representing the length in bytes of the password. Set to zero if no password is provided.

iComprLibraryLen
: Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in piComprLibrary. Set to zero if no library name is given.

db2Char data structure parameters:

pioData
: A pointer to a character data buffer. If NULL, no data will be returned.

iLength
: Input. The size of the pioData buffer.

oLength
: Output. The number of valid characters of data in the pioData buffer.

Usage notes:

This function is exempt from all label-based access control (LBAC) rules. It backs up all data, even protected data. Also, the data in the backup itself is not protected by LBAC. Any user with the backup and a place in which to restore it can gain access to the data.

Related tasks:

- "Using backup" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "BACKUP DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "BACKUP DATABASE " on page 437
- "db2Rollforward - Roll forward a database" on page 767
- "db2Restore - Restore a database or table space" on page 756
- "db2Recover - Restore and roll forward a database" on page 743

Related samples:

- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2CfgGet - Get the database manager or database configuration parameters

Returns the values of individual entries in a specific database configuration file or a database manager configuration file.

Scope:

Information about a specific database configuration file is returned only for the database partition on which it is executed.

Authorization:

None

Required connection:

To obtain the current online value of a configuration parameter for a specific database configuration file, a connection to the database is required. To obtain the current online value of a configuration parameter for the database manager, an instance attachment is required. Otherwise, a connection to a database or an attachment to an instance is not required.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2CfgGet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32 numItems;
   struct db2CfgParam *paramArray;
   db2Uint32 flags;
   char *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32 token;
   char *ptrvalue;
   db2Uint32 flags;
} db2CfgParam;

SQL_API_RC SQL_API_FN
  db2gCfgGet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32 numItems;
   struct db2gCfgParam *paramArray;
   db2Uint32 flags;
   db2Uint32 dbname_len;
   char *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32 token;
   db2Uint32 ptrvalue_len;
   char *ptrvalue;
   db2Uint32 flags;
} db2gCfgParam;
```

db2CfgGet API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
: Input. A pointer to the db2Cfg structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2Cfg data structure parameters:

numItems
: Input. The number of configuration parameters in the paramArray array. Set this value to db2CfgMaxParam to specify the largest number of elements in the paramArray.

paramArray
: Input. A pointer to the db2CfgParam structure.

flags
: Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

  db2CfgDatabase
  : Specifies to return the values in the database configuration file.

  db2CfgDatabaseManager
  : Specifies to return the values in the database manager configuration file.

  db2CfgImmediate
  : Returns the current values of the configuration parameters stored in memory.

  db2CfgDelayed
  : Gets the values of the configuration parameters on disk. These do not become the current values in memory until the next database connection or instance attachment.

  db2CfgGetDefaults
  : Returns the default values for the configuration parameter.

  db2CfgReset
  : Reset to default values.

dbname
: Input. The database name.

db2CfgParam data structure parameters:

token
: Input. The configuration parameter identifier.

  Valid entries and data types for the db2CfgParam token element are listed in ″Configuration parameters summary″.

ptrvalue
: Output. The configuration parameter value.

flags
: Input. Provides specific information for each parameter in a request. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgParamAutomatic
Indicates whether the retrieved parameter has a value of automatic. To determine whether a given configuration parameter has been set to automatic, perform a boolean AND operation against the value returned by the flag and the db2CfgParamAutomatic keyword defined in db2ApiDf.h.

db2gCfg data structure specific parameters:

dbname_len
Input. The length in bytes of dbname parameter.

db2gCfgParam data structure specific parameters:

ptrvalue_len
Input. The length in bytes of ptrvalue parameter.

Related tasks:
- “Configuring DB2 with configuration parameters” on page 1493

Related reference:
- “GET DATABASE CONFIGURATION ” on page 502
- “GET DATABASE MANAGER CONFIGURATION ” on page 508
- “DBCFG administrative view - Retrieve database configuration parameter information” in *Administrative SQL Routines and Views*
- “DBMCFG administrative view - Retrieve database manager configuration parameter information” in *Administrative SQL Routines and Views*
- “Configuration parameters summary” on page 1484
- “db2CfgSet - Set the database manager or database configuration parameters” on page 676
- “SQLCA data structure” in *Administrative API Reference*
- “db2AutoConfig API - Access the Configuration Advisor” in *Administrative API Reference*

Related samples:
- “dbinfo.c -- Set and get information at the database level (C)”
- “dbrecov.sqc -- How to recover a database (C)”
- “inauth.sqc -- How to display authorities at instance level (C)”
- “ininfo.c -- Set and get information at the instance level (C)”
- “tscreate.sqc -- How to create and drop buffer pools and table spaces (C)”
- “dbinfo.C -- Set and get information at the database level (C++)”
- “dbrecov.sqC -- How to recover a database (C++)”
- “inauth.sqC -- How to display authorities at instance level (C++)”
- “ininfo.C -- Set and get information at the instance level (C++)”
- “tscreate.sqC -- How to create and drop buffer pools and table spaces (C++)”

db2CfgSet - Set the database manager or database configuration parameters

Modifies individual entries in a specific database configuration file or a database manager configuration file. A database configuration file resides on every node on which the database has been created.

Scope:

Modifications to the database configuration file affect the node on which it is executed.

Authorization:

For modifications to the database configuration file, one of the following:
- sysadm
- sysctrl
- sysmaint

For modifications to the database manager configuration file:
- sysadm

Required connection:

To make an online modification of a configuration parameter for a specific database, a connection to the database is required. To make an online modification of a configuration parameter for the database manager, an attachment to an instance is not required.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2CfgSet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2Cfg
{
   db2Uint32 numItems;
   struct db2CfgParam *paramArray;
   db2Uint32 flags;
   char *dbname;
} db2Cfg;

typedef SQL_STRUCTURE db2CfgParam
{
   db2Uint32 token;
   char *ptrvalue;
   db2Uint32 flags;
} db2CfgParam;

SQL_API_RC SQL_API_FN
  db2gCfgSet (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gCfg
{
   db2Uint32 numItems;
   struct db2gCfgParam *paramArray;
   db2Uint32 flags;
   db2Uint32 dbname_len;
   char *dbname;
} db2gCfg;

typedef SQL_STRUCTURE db2gCfgParam
{
   db2Uint32 token;
   db2Uint32 ptrvalue_len;
   char *ptrvalue;
   db2Uint32 flags;
} db2gCfgParam;
```

db2CfgSet API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2Cfg structure.

pSqlca
Output. A pointer to the sqlca structure.

db2Cfg data structure parameters:

numItems
Input. The number of configuration parameters in the paramArray array. Set this value to db2CfgMaxParam to specify the largest number of elements in the paramArray.

paramArray
Input. A pointer to the db2CfgParam structure.

flags Input. Specifies the type of action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgDatabase
Specifies to return the values in the database configuration file.

db2CfgDatabaseManager
Specifies to return the values in the database manager configuration file.

db2CfgImmediate
Returns the current values of the configuration parameters stored in memory.

db2CfgDelayed
Gets the values of the configuration parameters on disk. These do not become the current values in memory until the next database connection or instance attachment.

db2CfgGetDefaults
Returns the default values for the configuration parameter.

db2CfgReset
Reset to default values.

dbname
Input. The database name.

db2CfgParam data structure parameters:

token Input. The configuration parameter identifier. Valid entries and data types for the db2CfgParam token element are listed in ″Configuration parameters summary″.

ptrvalue
Output. The configuration parameter value.

flags Input. Provides specific information for each parameter in a request. Valid values (defined in db2ApiDf header file, located in the include directory) are:

db2CfgParamAutomatic
Indicates whether the retrieved parameter has a value of automatic. To determine whether a given configuration parameter has been set to automatic, perform a boolean AND operation against the value returned by the flag and the db2CfgParamAutomatic keyword defined in db2ApiDf.h.

db2gCfg data structure specific parameters:

dbname_len
Input. The length in bytes of dbname parameter.

db2gCfgParam data structure specific parameters:

ptrvalue_len
Input. The length in bytes of ptrvalue parameter.

Related concepts:
- “Configuration parameters” on page 1475

Related tasks:
- “Configuring DB2 with configuration parameters” on page 1493
- “Changing node and database configuration files” in *Administration Guide: Implementation*

Related reference:
- “AUTOCONFIGURE command” in *Command Reference*
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION ” on page 627
- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629
- “ADMIN_CMD procedure ÔÇô Run administrative commands” in *Administrative SQL Routines and Views*
- “Configuration parameters summary” on page 1484
- “db2CfgGet - Get the database manager or database configuration parameters” on page 672
- “SQLCA data structure” in *Administrative API Reference*
- “RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*

- “UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2AutoConfig API - Access the Configuration Advisor” in *Administrative API Reference*

Related samples:

- “dbinfo.c -- Set and get information at the database level (C)”
- “dbrecov.sqc -- How to recover a database (C)”
- “ininfo.c -- Set and get information at the instance level (C)”
- “dbinfo.C -- Set and get information at the database level (C++)”
- “dbrecov.sqC -- How to recover a database (C++)”
- “ininfo.C -- Set and get information at the instance level (C++)”

db2DatabaseRestart - Restart database

Restarts a database that has been abnormally terminated and left in an inconsistent state. At the successful completion of this API, the application remains connected to the database if the user has CONNECT privilege.

Scope:

This API affects only the database partition server on which it is executed.

Authorization:

None

Required connection:

This API establishes a database connection.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2DatabaseRestart (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2RestartDbStruct
{
   char *piDatabaseName;
   char *piUserId;
   char *piPassword;
   char *piTablespaceNames;
   db2int32 iOption;
} db2RestartDbStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseRestart (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2gRestartDbStruct
{
   db2Uint32 iDatabaseNameLen;
   db2Uint32 iUserIdLen;
   db2Uint32 iPasswordLen;
   db2Uint32 iTablespaceNamesLen;
   char *piDatabaseName;
   char *piUserId;
   char *piPassword;
   char *piTablespaceNames;
} db2gRestartDbStruct;
```

db2DatabaseRestart API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, pParamStruct.

pParamStruct
: Input. A pointer to the db2RestartDbStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2RestartDbStruct data structure parameters:

piDatabaseName
: Input. A pointer to a string containing the alias of the database that is to be restarted.

piUserId
: Input. A pointer to a string containing the user name of the application. May be NULL.

piPassword
: Input. A pointer to a string containing a password for the specified user name (if any). May be NULL.

piTablespaceNames
: Input. A pointer to a string containing a list of table space names to be dropped during the restart operation. May be NULL.

iOption
: Input. Valid values are:

  DB2_DB_SUSPEND_NONE
  : Performs normal crash recovery.

  DB2_DB_RESUME_WRITE
  : Required to perform crash recovery on a database that has I/O writes suspended.

db2gRestartDbStruct data structure specific parameters:

iDatabaseNameLen
: Input. Length in bytes of piDatabaseName parameter.

iUserIdLen
: Input. Length in bytes of piUserId parameter.

iPasswordLen
: Input. Length in bytes of piPassword parameter.

iTablespaceNamesLen
Input. Length in bytes of piTablespaceNames parameter.

Usage notes:

Call this API if an attempt to connect to a database returns an error message, indicating that the database must be restarted. This action occurs only if the previous session with this database terminated abnormally (due to power failure, for example).

At the completion of this API, a shared connection to the database is maintained if the user has CONNECT privilege, and an SQL warning is issued if any indoubt transactions exist. In this case, the database is still usable, but if the indoubt transactions are not resolved before the last connection to the database is dropped, another call to the API must be completed before the database can be used again.

In the case of circular logging, a database restart operation will fail if there is any problem with the table spaces, such as an I/O error, an unmounted file system, and so on. If losing such table spaces is not an issue, their names can be explicitly specified; this will put them into drop pending state, and the restart operation can complete successfully.

REXX API syntax:

```
RESTART DATABASE database_alias [USER username USING password]
```

REXX API parameters:

database_alias
Alias of the database to be restarted.

username
User name under which the database is to be restarted.

password
Password used to authenticate the user name.

Related reference:

- "RESTART DATABASE " on page 589
- "SQLCA data structure" in *Administrative API Reference*

Related samples:

- "dbconn.sqc -- How to connect to and disconnect from a database (C)"
- "dbconn.sqC -- How to connect to and disconnect from a database (C++)"

db2DatabaseQuiesce - Quiesce the database

Forces all users off the database, immediately rolls back all active transactions or waits for them to complete their current units of work within the number of minutes specified (if they cannot be completed within the specified number of minutes, the operation will fail), and puts the database into quiesce mode. This API provides exclusive access to the database. During this quiesced period, system administration can be performed on the database by users with appropriate authority. After administration is complete, you can unquiesce the database, using the db2DatabaseUnquiesce API. The db2DatabaseUnquiesce API allows other users to connect to the database, without having to shut down and perform another database start. In this mode only groups or users with QUIESCE CONNECT authority and sysadm, sysmaint, or sysctrl will have access to the database and its objects.

Authorization:

One of the following:
- sysadm
- dbadm

Required connection:

Database

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2DatabaseQuiesce (
            db2Uint32 versionNumber,
            void * pParmStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbQuiesceStruct
{
  char *piDatabaseName;
  db2Uint32 iImmediate;
  db2Uint32 iForce;
  db2Uint32 iTimeout;
} db2DbQuiesceStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseQuiesce (
            db2Uint32 versionNumber,
            void * pParmStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbQuiesceStruct
{
  db2Uint32 iDatabaseNameLen;
  char *piDatabaseName;
  db2Uint32 iImmediate;
  db2Uint32 iForce;
  db2Uint32 iTimeout;
} db2gDbQuiesceStruct;
```

db2DatabaseQuiesce API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
: Input. A pointer to the db2DbQuiesceStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2DbQuiesceStruct data structure parameters:

piDatabaseName
Input. The database name.

iImmediate
Input. Valid values are:

TRUE=1
Force the applications immediately.

FALSE=0
Deferred force. Applications will wait the number of minutes specified by iTimeout parameter to let their current units of work be completed, and then will terminate. If this deferred force cannot be completed within the number of minutes specified by iTimeout parameter, the quiesce operation will fail.

iForce Input. Reserved for future use.

iTimeout
Input. Specifies the time, in minutes, to wait for applications to commit the current unit of work. If iTimeout is not specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the start_stop_timeout database manager configuration parameter will be used.

db2gDbQuiesceStruct data structure specific parameters:

iDatabaseNameLen
Input. Specifies the length in bytes of piDatabaseName.

Related reference:

- "QUIESCE " on page 567
- "db2DatabaseUnquiesce - Unquiesce database" on page 683
- "SQLCA data structure" in *Administrative API Reference*
- "QUIESCE DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

db2DatabaseUnquiesce - Unquiesce database

Restores user access to databases which have been quiesced for maintenance or other reasons. User access is restored without necessitating a shutdown and database restart.

Authorization:

One of the following:

- sysadm
- dbadm

Required connection:

Database

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2DatabaseUnquiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2DbUnquiesceStruct
{
             char *piDatabaseName;
} db2DbUnquiesceStruct;

SQL_API_RC SQL_API_FN
  db2gDatabaseUnquiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gDbUnquiesceStruct
{
             db2Uint32 iDatabaseNameLen;
             char *piDatabaseName;
} db2gDbUnquiesceStruct;
```

db2DatabaseUnquiesce API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2DbUnquiesceStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2DbUnquiesceStruct data structure parameters:

piDatabaseName
Input. The database name.

db2gDbUnquiesceStruct data structure specific parameters:

iDatabaseNameLen
Input. Specifies the length in bytes of piDatabaseName.

Related reference:

- "UNQUIESCE " on page 626
- "db2DatabaseQuiesce - Quiesce the database" on page 681
- "SQLCA data structure" in *Administrative API Reference*
- "UNQUIESCE DATABASE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

db2Export - Export data from a database

Exports data from a database to one of several external file formats. The user specifies the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Authorization:

One of the following:
- sysadm
- dbadm

or CONTROL or SELECT privilege on each participating table or view. Label-based access control (LBAC) is enforced for this function. The data that is exported may be limited by the LBAC credentials of the caller if the data is protected by LBAC.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Export (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ExportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqlu_media_list *piLobFileList;
   struct sqldcol *piDataDescriptor;
   struct sqllob *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ExportOut *poExportInfoOut;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2ExportStruct;

typedef SQL_STRUCTURE db2ExportIn
{
   db2Uint16 *piXmlSaveSchema;
} db2ExportIn;

typedef SQL_STRUCTURE db2ExportOut
{
   db2Uint64 oRowsExported;
} db2ExportOut;

SQL_API_RC SQL_API_FN
  db2gExport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gExportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqlu_media_list *piLobFileList;
   struct sqldcol *piDataDescriptor;
   struct sqllob *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ExportOut *poExportInfoOut;
   db2Uint16 iDataFileNameLen;
   db2Uint16 iFileTypeLen;
   db2Uint16 iMsgFileNameLen;
   struct db2ExportIn *piExportInfoIn;
   struct sqlu_media_list *piXmlPathList;
   struct sqlu_media_list *piXmlFileList;
} db2gExportStruct;
```

db2Export API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2ExportStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ExportStruct data structure parameters:

piDataFileName
Input. A string containing the path and the name of the external file into which the data is to be exported.

piLobPathList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files are to be stored. Exported LOB data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piLobFileList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its sqlu_location_entry structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on. When creating LOB files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piLobPathList), and then appending a 3-digit sequence number and the .lob extension. For example, if the current LOB path is the directory /u/foo/lob/path, the current LOB file name is bar, and the LOBSINSEPFILES file type modifier is set, then the created LOB files will be /u/foo/LOB/path/bar.001.lob, /u/foo/LOB/path/bar.002.lob, and so on. If the LOBSINSEPFILES file type modifier is not set, then all the LOB documents will be concatenated and put into one file /u/foo/lob/path/bar.001.lob

piDataDescriptor
Input. Pointer to an sqldcol structure specifying the column names for the output file. The value of the dcolmeth field determines how the remainder of the information provided in this parameter is interpreted by the export utility. Valid values for this parameter (defined in sqlutil header file, located in the include directory) are:

SQL_METH_N
Names. Specify column names to be used in the output file.

SQL_METH_D
Default. Existing column names from the table are to be used in the output file. In this case, the number of columns and the column specification array are both ignored. The column names are derived from the output of the SELECT statement specified in pActionString.

piActionString
Input. Pointer to an sqllob structure containing a valid dynamic SQL SELECT statement. The structure contains a 4-byte long field, followed by the characters that make up the SELECT statement. The SELECT statement specifies the data to be extracted from the database and written to the external file.

The columns for the external file (from piDataDescriptor), and the database columns from the SELECT statement, are matched according to their respective list/structure positions. The first column of data selected from the database is placed in the first column of the external file, and its column name is taken from the first element of the external column array.

piFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats (defined in sqlutil header file) are:

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs.

SQL_IXF
PC version of the Integrated Exchange Format, the preferred method for exporting data from a table. Data exported to this file format can later be imported or loaded into the same table or into another database manager table.

piFileTypeMod
Input. A pointer to an sqldcol structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link below: ″File type modifiers for the export utility.″

piMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the name of an operating system file or a standard device. If the file already exists, the information is appended . If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API. If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested export operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

poExportInfoOut
A pointer to the db2ExportOut structure.

piExportInfoIn
Input. Pointer to the db2ExportIn structure.

piXmlPathList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files are to be stored. Exported XML data will be distributed evenly among all the paths listed in the sqlu_media_entry structure.

piXmlFileList
Input. Pointer to an sqlu_media_list structure with its media_type field set to SQLU_CLIENT_LOCATION, and its sqlu_location_entry structure containing base file names.

When the name space is exhausted using the first name in this list, the API will use the second name, and so on. When creating XML files during an export operation, file names are constructed by appending the current base name from this list to the current path (from piXmlFileList), and then appending a 3-digit sequence number and the .xml extension. For example, if the current XML path is the directory /u/foo/xml/path, the current XML file name is bar, and the XMLINSEPFILES file type modifier is set, then the created XML files will be /u/foo/xml/path/bar.001.xml, /u/foo/xml/path/bar.002.xml, and so on. If the XMLINSEPFILES file type modifier is not set, then all the XML documents will be concatenated and put into one file /u/foo/xml/path/bar.001.xml

db2ExportIn data structure parameters:

piXmlSaveSchema
Input. Indicates that the SQL identifier of the XML schema used to validate each exported XML document should be saved in the exported data file. Possible values are TRUE and FALSE.

db2ExportOut data structure parameters:

oRowsExported
Output. Returns the number of records exported to the target file.

db2gExportStruct data structure specific parameters:

iDataFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the data file name.

iFileTypeLen
Input. A 2-byte unsigned integer representing the length in bytes of the file type.

iMsgFileNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the message file name.

Usage notes:

Before starting an export operation, you must complete all table operations and release all locks in one of two ways:

- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

Table aliases can be used in the SELECT statement.

The messages placed in the message file include the information returned from the message retrieval service. Each message begins on a new line.

If the export utility produces warnings, the message will be written out to a message file, or standard output if one is not specified.

A warning message is issued if the number of columns (dcolnum field of sqldcol structure) in the external column name array, piDataDescriptor, is not equal to the number of columns generated by the SELECT statement. In this case, the number of columns written to the external file is the lesser of the two numbers. Excess database columns or external column names are not used to generate the output file.

If the db2uexpm.bnd module or any other shipped .bnd files are bound manually, the format option on the binder must not be used.

DB2 Connect can be used to export tables from DRDA servers such as DB2 for z/OS and OS/390, DB2 for VM and VSE, and DB2 for iSeries. Only PC/IXF export is supported.

PC/IXF import should be used to move data between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The export utility will not create multiple-part PC/IXF files when invoked from an AIX system.

Index definitions for a table are included in the PC/IXF file when the contents of a single database table are exported to a PC/IXF file with a pActionString parameter beginning with SELECT * FROM tablename, and the piDataDescriptor parameter specifying default names. Indexes are not saved for views, or if the SELECT clause of the piActionString includes a join. A WHERE clause, a GROUP BY clause, or a HAVING clause in the piActionString parameter will not prevent the saving of indexes. In all of these cases, when exporting from typed tables, the entire hierarchy must be exported.

The export utility will store the NOT NULL WITH DEFAULT attribute of the table in an IXF file if the SELECT statement provided is in the form: SELECT * FROM tablename.

When exporting typed tables, subselect statements can only be expressed by specifying 7the target table name and the WHERE clause. Fullselect and select-statement cannot be specified when exporting a hierarchy.

For file formats other than IXF, it is recommended that the traversal order list be specified, because it tells DB2 how to traverse the hierarchy, and what sub-tables to export. If this list is not specified, all tables in the hierarchy are exported, and the default order is the OUTER order. The alternative is to use the default order, which is the order given by the OUTER function.

Note: Use the same traverse order during an import operation. The load utility does not support loading hierarchies or sub-hierarchies.

DB2 Data Links Manager considerations:

To ensure that a consistent copy of the table and the corresponding files referenced by the DATALINK columns are copied for export, do the following:

1. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename SHARE. This ensures that no update transactions are in progress when EXPORT is run.
2. Issue the EXPORT command.
3. Run the dlfm_export utility at each Data Links server. Input to the dlfm_export utility is the control file name, which is generated by the export utility. This produces a tar (or equivalent) archive of the files listed within the control file. dlfm_export does not capture the ACLs information of the files that are archived.
4. Issue the command: QUIESCE TABLESPACES FOR TABLE tablename RESET.

   This makes the table available for updates.

EXPORT is executed as an SQL application. The rows and columns satisfying the SELECT statement conditions are extracted from the database. For the DATALINK columns, the SELECT statement should not specify any scalar function.

Successful execution of EXPORT results in generation of the following files:

- An export data file as specified in the EXPORT command. A DATALINK column value in this file has the same format as that used by the IMPORT and LOAD utilities. When the DATALINK column value is the SQL NULL value, handling is the same as that for other data types.
- Control files server_name, which are generated for each Data Links server. On the Windows operating system, a single control file, ctrlfile.lst, is used by all Data Links servers. These control files are placed in the directory <data-file path>/dlfm/YYYYMMDD/HHMMSS (on the Windows operating system,

ctrlfile.lst is placed in the directory <data-file path>\dlfm\YYYYMMDD\HHMMSS). YYYYMMDD represents the date (year month day), and HHMMSS represents the time (hour minute second).

REXX API syntax:

```
EXPORT :stmt TO datafile OF filetype
[MODIFIED BY :filetmod] [USING :dcoldata]
MESSAGES msgfile [ROWS EXPORTED :number]


CONTINUE EXPORT


STOP EXPORT
```

REXX API parameters:

stmt
A REXX host variable containing a valid dynamic SQL SELECT statement. The statement specifies the data to be extracted from the database.

datafile
Name of the file into which the data is to be exported.

filetype
The format of the data in the export file. The supported file formats are:

DEL	Delimited ASCII
WSF	Worksheet format
IXF	PC version of Integrated Exchange Format.

filetmod
A host variable containing additional processing options.

dcoldata
A compound REXX host variable containing the column names to be used in the export file. In the following, XXX represents the name of the host variable:

XXX.0	Number of columns (number of elements in the remainder of the variable).
XXX.1	First column name.
XXX.2	Second column name.
XXX.3	and so on.

If this parameter is NULL, or a value for dcoldata has not been specified, the utility uses the column names from the database table.

msgfile
File, path, or device name where error and warning messages are to be sent.

number
A host variable that will contain the number of exported rows.

Related tasks:

- "Exporting data" in *Data Movement Utilities Guide and Reference*

Related reference:

- "sqlchar data structure" in *Administrative API Reference*

- "sqldcol data structure" in *Administrative API Reference*
- "sqllob data structure" in *Administrative API Reference*
- "sqlu_media_list data structure" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "ADMIN_CMD procedure ÔÇô Run administrative commands" in *Administrative SQL Routines and Views*
- "EXPORT " on page 496
- "EXPORT command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 692
- "db2Load - Load data into a table" on page 723

Related samples:

- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tload.sqb -- How to export and load table data (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2Import - Import data into a table, hierarchy, nickname or view

Inserts data from an external file with a supported file format into a table, hierarchy, nickname or view. The load utility is faster than this function. The load utility, however, does not support loading data at the hierarchy level or loading into a nickname.

Authorization:

- IMPORT using the INSERT option requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on each participating table, view or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on the table, view or nickname
  - INSERT, SELECT, UPDATE and DELETE privilege on each participating table or view
- IMPORT to an existing table using the REPLACE or REPLACE_CREATE option, requires one of the following:
  - sysadm
  - dbadm
  - CONTROL privilege on the table or view
  - INSERT, SELECT, and DELETE privilege on the table or view
- IMPORT to a new table using the CREATE or REPLACE_CREATE option, requires one of the following:

  – sysadm
  – dbadm
  – CREATETAB authority on the database and USE privilege on the table space, as well as one of:
    - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
    - CREATIN privilege on the schema, if the schema name of the table refers to an existing schema
- IMPORT to a table or a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  – sysadm
  – dbadm
  – CREATETAB authority on the database, and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name of the table does not exist
    - CREATEIN privilege on the schema, if the schema of the table exists
    - CONTROL privilege on every sub-table in the hierarchy, if the REPLACE_CREATE option on the entire hierarchy is used
- IMPORT to an existing hierarchy using the REPLACE option requires one of the following:
  – sysadm
  – dbadm
  – CONTROL privilege on every sub-table in the hierarchy

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established.

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Import (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ImportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2ImportIn *piImportInfoIn;
   struct db2ImportOut *poImportInfoOut;
   db2int32 *piNullIndicators;
   struct sqlu_media_list *piXmlPathList;
} db2ImportStruct;

typedef SQL_STRUCTURE db2ImportIn
```

```
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2ImportIn;

typedef SQL_STRUCTURE db2ImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2ImportOut;

typedef SQL_STRUCTURE db2DMUXmlMapSchema
{
   struct db2Char                      iMapFromSchema;
   struct db2Char                      iMapToSchema;
} db2DMUXmlMapSchema;

typedef SQL_STRUCTURE db2DMUXmlValidateXds
{
   struct db2Char *piDefaultSchema;
   db2Uint32 iNumIgnoreSchemas;
   struct db2Char *piIgnoreSchemas;
   db2Uint32 iNumMapSchemas;
   struct db2DMUXmlMapSchema *piMapSchemas;
} db2DMUXmlValidateXds;

typedef SQL_STRUCTURE db2DMUXmlValidateSchema
{
   struct db2Char *piSchema;
} db2DMUXmlValidateSchema;

typedef SQL_STRUCTURE db2DMUXmlValidate
{
   db2Uint16 iUsing;
   struct db2DMUXmlValidateXds *piXdsArgs;
   struct db2DMUXmlValidateSchema *piSchemaArgs;
} db2DMUXmlValidate;

SQL_API_RC SQL_API_FN
  db2gImport (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gImportStruct
{
   char *piDataFileName;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piMsgFileName;
   db2int16 iCallerAction;
   struct db2gImportIn *piImportInfoIn;
   struct dbg2ImportOut *poImportInfoOut;
   db2int32 *piNullIndicators;
   db2Uint16 iDataFileNameLen;
   db2Uint16 iFileTypeLen;
   db2Uint16 iMsgFileNameLen;
   struct sqlu_media_list *piXmlPathList;
} db2gImportStruct;

typedef SQL_STRUCTURE db2gImportIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   db2Uint64 iSkipcount;
   db2int32 *piCommitcount;
   db2Uint32 iWarningcount;
   db2Uint16 iNoTimeout;
   db2Uint16 iAccessLevel;
   db2Uint16 *piXmlParse;
   struct db2DMUXmlValidate *piXmlValidate;
} db2gImportIn;

typedef SQL_STRUCTURE db2gImportOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsInserted;
   db2Uint64 oRowsUpdated;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsCommitted;
} db2gImportOut;
```

db2Import API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed in as the second parameter pParmStruct.

pParmStruct
Input/Output. A pointer to the db2ImportStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ImportStruct data structure parameters:

piDataFileName
Input. A string containing the path and the name of the external input file from which the data is to be imported.

piLobPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the LOB files can be found. This parameter is not valid when you import to a nickname.

piDataDescriptor
Input. Pointer to an sqldcol structure containing information about the columns being selected for import from the external file. The value of the dcolmeth field determines how the remainder of the information provided in this parameter is interpreted by the import utility. Valid values for this parameter are:

SQL_METH_N
Names. Selection of columns from the external input file is by column name.

SQL_METH_P
Positions. Selection of columns from the external input file is by column position.

SQL_METH_L
Locations. Selection of columns from the external input file is by column location. The database manager rejects an import call with a location pair that is invalid because of any one of the following conditions:

- Either the beginning or the ending location is not in the range from 1 to the largest signed 2-byte integer.
- The ending location is smaller than the beginning location.
- The input column width defined by the location pair is not compatible with the type and the length of the target column.

A location pair with both locations equal to zero indicates that a nullable column is to be filled with NULLs.

SQL_METH_D
Default. If piDataDescriptor is NULL, or is set to SQL_METH_D, default selection of columns from the external input file is done. In this case, the number of columns and the column specification array are both ignored. For DEL, IXF, or WSF files, the first n columns of data in the external input file are taken in their natural order, where n is the number of database columns into which the data is to be imported.

piActionString
Input. Pointer to an sqlchar structure containing a 2-byte long field, followed by an array of characters identifying the columns into which data is to be imported.

The character array is of the form:

```
{INSERT | INSERT_UPDATE | REPLACE | CREATE | REPLACE_CREATE}
INTO {tname[(tcolumn-list)] |
[{ALL TABLES | (tname[(tcolumn-list)][, tname[(tcolumn-list)]])}]
[IN] HIERARCHY {STARTING tname | (tname[, tname])}
[UNDER sub-table-name | AS ROOT TABLE]}
[DATALINK SPECIFICATION datalink-spec]
```

INSERT
Adds the imported data to the table without changing the existing table data.

INSERT_UPDATE
Adds the imported rows if their primary key values are not in the table, and uses them for update if their primary key values are found. This option is only valid if the target table has a primary key, and the specified (or implied) list of target columns being imported includes all columns for the primary key. This option cannot be applied to views.

REPLACE
Deletes all existing data from the table by truncating the table object, and inserts the imported data. The table definition and the index definitions are not changed. (Indexes are deleted and replaced if indexixf is in FileTypeMod, and FileType is SQL_IXF.) If the table is not already defined, an error is returned.

Note: If an error occurs after the existing data is deleted, that data is lost.

This parameter is not valid when you import to a nickname.

CREATE
Creates the table definition and the row contents using the information in the specified PC/IXF file, if the specified table is not defined. If the file was previously exported by DB2, indexes are also created. If the specified table is already defined, an error is returned. This option is valid for the PC/IXF file format only. This parameter is not valid when you import to a nickname.

REPLACE_CREATE
Replaces the table contents using the PC/IXF row information in the PC/IXF file, if the specified table is defined. If the table is not already defined, the table definition and row contents are created using the information in the specified PC/IXF file. If the PC/IXF file was previously exported by DB2, indexes are also created. This option is valid for the PC/IXF file format only.

Note: If an error occurs after the existing data is deleted, that data is lost.

This parameter is not valid when you import to a nickname.

tname The name of the table, typed table, view, or object view into which the data is to be inserted. An alias for REPLACE, INSERT_UPDATE, or INSERT can be specified, except in the case of a down-level server, when a qualified or unqualified name should be specified. If it is a view, it cannot be a read-only view.

tcolumn-list
A list of table or view column names into which the data is to be inserted. The column names must be separated by commas. If column names are not specified, column names as defined in the CREATE TABLE or the ALTER TABLE statement are used. If no column list is specified for typed tables, data is inserted into all columns within each sub-table.

sub-table-name
Specifies a parent table when creating one or more sub-tables under the CREATE option.

ALL TABLES
An implicit keyword for hierarchy only. When importing a hierarchy, the default is to import all tables specified in the traversal-order-list.

HIERARCHY
Specifies that hierarchical data is to be imported.

STARTING
Keyword for hierarchy only. Specifies that the default order, starting from a given sub-table name, is to be used.

UNDER
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created under a given sub-table.

AS ROOT TABLE
Keyword for hierarchy and CREATE only. Specifies that the new hierarchy, sub-hierarchy, or sub-table is to be created as a stand-alone hierarchy.

DATALINK SPECIFICATION datalink-spec
Specifies parameters pertaining to DB2 Data Links Manager. These parameters can be specified using the same syntax as in the IMPORT command.

The tname and the tcolumn-list parameters correspond to the tablename and the colname lists of SQL INSERT statements, and have the same restrictions.

The columns in tcolumn-list and the external columns (either specified or implied) are matched according to their position in the list or the structure (data from the first column specified in the sqldcol structure is inserted into the table or view field corresponding to the first element of the tcolumn-list).

If unequal numbers of columns are specified, the number of columns actually processed is the lesser of the two numbers. This could result in an error (because there are no values to place in some non-nullable table fields) or an informational message (because some external file columns are ignored).

This parameter is not valid when you import to a nickname.

piFileType
Input. A string that indicates the format of the data within the external file. Supported external file formats are:

SQL_ASC
Non-delimited ASCII.

SQL_DEL
Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

SQL_IXF
PC version of the Integrated Exchange Format, the preferred method for exporting data from a table so that it can be imported later into the same table or into another database manager table.

SQL_WSF
Worksheet formats for exchange with Lotus Symphony and 1-2-3 programs. The WSF file type is not supported when you import to a nickname.

piFileTypeMod
Input. A pointer to a structure containing a 2-byte long field, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

Not all options can be used with all of the supported file types. See related link "File type modifiers for the import utility".

piMsgFileName
Input. A string containing the destination for error, warning, and informational messages returned by the utility. It can be the path and the

name of an operating system file or a standard device. If the file already exists, it is appended to. If it does not exist, a file is created.

iCallerAction
Input. An action requested by the caller. Valid values are:

SQLU_INITIAL
Initial call. This value must be used on the first call to the API. If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested import operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape condition). It specifies that the user action requested by the utility was not performed, and the utility is to terminate processing the initial request.

piImportInfoIn
Input. Pointer to the db2ImportIn structure.

poImportInfoOut
Output. Pointer to the db2ImportOut structure.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. The number of elements in this array must match the number of columns in the input file; there is a one-to-one ordered correspondence between the elements of this array and the columns being imported from the data file. Therefore, the number of elements must equal the dcolnum field of the piDataDescriptor parameter. Each element of the array contains a number identifying a column in the data file that is to be used as a null indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified column in the data file must contain a Y or an N. A Y indicates that the table column data is NULL, and N indicates that the table column data is not NULL.

piXmlPathList
Input. Pointer to an sqlu_media_list with its media_type field set to SQLU_LOCAL_MEDIA, and its sqlu_media_entry structure listing paths on the client where the XML files can be found.

db2ImportIn data structure parameters:

iRowcount
Input. The number of physical records to be loaded. Allows a user to load only the first iRowcount rows in a file. If iRowcount is 0, import will attempt to process all the rows from the file.

iRestartcount
: Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iSkipcount parameter. iRestartcount and iSkipcount parameters are mutually exclusive.

iSkipcount
: Input. The number of records to skip before starting to insert or update records. Functionally equivalent to iRestartcount.

piCommitcount
: Input. The number of records to import before committing them to the database. A commit is performed whenever piCommitcount records are imported. A NULL value specifies the default commit count value, which is zero for offline import and AUTOMATIC for online import. Commitcount AUTOMATIC is specified by passing in the value DB2IMPORT_COMMIT_AUTO.

iWarningcount
: Input. Stops the import operation after iWarningcount warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the import file or the target table is specified incorrectly, the import utility will generate a warning for each row that it attempts to import, which will cause the import to fail.

If iWarningcount is 0, or this option is not specified, the import operation will continue regardless of the number of warnings issued.

iNoTimeout
: Input. Specifies that the import utility will not time out while waiting for locks. This option supersedes the locktimeout database configuration parameter. Other applications are not affected. Valid values are:

DB2IMPORT_LOCKTIMEOUT
: Indicates that the value of the locktimeout configuration parameter is respected.

DB2IMPORT_NO_LOCKTIMEOUT
: Indicates there is no timeout.

iAccessLevel
: Input. Specifies the access level. Valid values are:

- SQLU_ALLOW_NO_ACCESS
: Specifies that the import utility locks the table exclusively.

- SQLU_ALLOW_WRITE_ACCESS
: Specifies that the data in the table should still be accessible to readers and writers while the import is in progress.

An intent exclusive (IX) lock on the target table is acquired when the first row is inserted. This allows concurrent readers and writers to access table data. Online mode is not compatible with the REPLACE, CREATE, or REPLACE_CREATE import options. Online mode is not supported in conjunction with buffered inserts. The import operation will periodically commit inserted data to prevent lock escalation to a table lock and to avoid running out of active log space. These commits will be performed even if the piCommitCount parameter was not used. During each commit, import will lose its IX table lock, and will attempt to reacquire it after the commit. This parameter is required when you import to a nickname and piCommitCount parameter must be specified with a valid number (AUTOMATIC is not considered a valid option).

piXmlParse
Input. Type of parsing that should occur for XML documents. Valid values found in the db2ApiDf header file in the include directory, are:

DB2DMU_XMLPARSE_PRESERVE_WS
Whitespace should be preserved.

DB2DMU_XMLPARSE_STRIP_WS
Whitespace should be stripped.

piXmlValidate
Input. Pointer to the db2DMUXmlValidate structure. Indicates that XML schema validation should occur for XML documents.

db2ImportOut data structure parameters:

oRowsRead
Output. Number of records read from the file during import.

oRowsSkipped
Output. Number of records skipped before inserting or updating begins.

oRowsInserted
Output. Number of rows inserted into the target table.

oRowsUpdated
Output. Number of rows in the target table updated with information from the imported records (records whose primary key value already exists in the table).

oRowsRejected
Output. Number of records that could not be imported.

oRowsCommitted
Output. Number of records imported successfully and committed to the database.

db2DMUXmlMapSchema data structure parameters:

iMapFromSchema
Input. The SQL identifier of the XML schema to map from.

iMapToSchema
Input. The SQL identifier of the XML schema to map to.

db2DMUXmlValidateXds data structure parameters:

piDefaultSchema
Input. The SQL identifier of the XML schema that should be used for validation when an XDS does not contain an SCH attribute.

iNumIgnoreSchemas
Input. The number of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

piIgnoreSchemas
Input. The list of XML schemas that will be ignored during XML schema validation if they are referred to by an SCH attribute in XDS.

iNumMapSchemas
Input. The number of XML schemas that will be mapped during XML schema validation. The first schema in the schema map pair represents a

schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

piMapSchemas
Input. The list of XML schema pairs, where each pair represents a mapping of one schema to a different one. The first schema in the pair represents a schema that is referred to by an SCH attribute in an XDS. The second schema in the pair represents the schema that should be used to perform schema validation.

db2DMUXmlValidateSchema data structure parameters:

piSchema
Input. The SQL identifier of the XML schema to use.

db2DMUXmlValidate data structure parameters:

iUsing
Input. A specification of what to use to perform XML schema validation. Valid values found in the db2ApiDf header file in the include directory, are:

- DB2DMU_XMLVAL_XDS
Validation should occur according to the XDS. This corresponds to the CLP ″XMLVALIDATE USING XDS″ clause.

- DB2DMU_XMLVAL_SCHEMA
Validation should occur according to a specified schema. This corresponds to the CLP ″XMLVALIDATE USING SCHEMA″ clause.

- DB2DMU_XMLVAL_SCHEMALOC_HINTS
Validation should occur according to schemaLocation hints found within the XML document. This corresponds to the ″XMLVALIDATE USING SCHEMALOCATION HINTS″ clause.

piXdsArgs
Input. Pointer to a db2DMUXmlValidateXds structure, representing arguments that correspond to the CLP ″XMLVALIDATE USING XDS″ clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_XDS.

piSchemaArgs
Input. Pointer to a db2DMUXmlValidateSchema structure, representing arguments that correspond to the CLP ″XMLVALIDATE USING SCHEMA″ clause.

This parameter applies only when the iUsing parameter in the same structure is set to DB2DMU_XMLVAL_SCHEMA.

db2gImportStruct data structure specific parameters:

iDataFileNameLen
Input. Specifies the length in bytes of piDataFileName parameter.

iFileTypeLen
Input. Specifies the length in bytes of piFileType parameter.

iMsgFileNameLen
Input. Specifies the length in bytes of piMsgFileName parameter.

Usage notes:

Before starting an import operation, you must complete all table operations and release all locks in one of two ways:
- Close all open cursors that were defined with the WITH HOLD clause, and commit the data changes by executing the COMMIT statement.
- Roll back the data changes by executing the ROLLBACK statement.

The import utility adds rows to the target table using the SQL INSERT statement.

The utility issues one INSERT statement for each row of data in the input file. If an INSERT statement fails, one of two actions result:
- If it is likely that subsequent INSERT statements can be successful, a warning message is written to the message file, and processing continues.
- If it is likely that subsequent INSERT statements will fail, and there is potential for database damage, an error message is written to the message file, and processing halts.

The utility performs an automatic COMMIT after the old rows are deleted during a REPLACE or a REPLACE_CREATE operation. Therefore, if the system fails, or the application interrupts the database manager after the table object is truncated, all of the old data is lost. Ensure that the old data is no longer needed before using these options.

If the log becomes full during a CREATE, REPLACE, or REPLACE_CREATE operation, the utility performs an automatic COMMIT on inserted records. If the system fails, or the application interrupts the database manager after an automatic COMMIT, a table with partial data remains in the database. Use the REPLACE or the REPLACE_CREATE option to rerun the whole import operation, or use INSERT with the iRestartcount parameter set to the number of rows successfully imported.

By default, automatic COMMITs are not performed for the INSERT or the INSERT_UPDATE option. They are, however, performed if the *piCommitcount parameter is not zero. A full log results in a ROLLBACK.

Whenever the import utility performs a COMMIT, two messages are written to the message file: one indicates the number of records to be committed, and the other is written after a successful COMMIT. When restarting the import operation after a failure, specify the number of records to skip, as determined from the last successful COMMIT.

The import utility accepts input data with minor incompatibility problems (for example, character data can be imported using padding or truncation, and numeric data can be imported with a different numeric data type), but data with major incompatibility problems is not accepted.

One cannot REPLACE or REPLACE_CREATE an object table if it has any dependents other than itself, or an object view if its base table has any dependents (including itself). To replace such a table or a view, do the following:
1. Drop all foreign keys in which the table is a parent.
2. Run the import utility.
3. Alter the table to recreate the foreign keys.

If an error occurs while recreating the foreign keys, modify the data to maintain referential integrity.

Referential constraints and foreign key definitions are not preserved when creating tables from PC/IXF files. (Primary key definitions are preserved if the data was previously exported using SELECT *.)

Importing to a remote database requires enough disk space on the server for a copy of the input data file, the output message file, and potential growth in the size of the database.

If an import operation is run against a remote database, and the output message file is very long (more than 60 KB), the message file returned to the user on the client may be missing messages from the middle of the import operation. The first 30 KB of message information and the last 30 KB of message information are always retained.

Importing PC/IXF files to a remote database is much faster if the PC/IXF file is on a hard drive rather than on diskettes. Non-default values for piDataDescriptor, or specifying an explicit list of table columns in piActionString, makes importing to a remote database slower.

The database table or hierarchy must exist before data in the ASC, DEL, or WSF file formats can be imported; however, if the table does not already exist, IMPORT CREATE or IMPORT REPLACE_CREATE creates the table when it imports data from a PC/IXF file. For typed tables, IMPORT CREATE can create the type hierarchy and the table hierarchy as well.

PC/IXF import should be used to move data (including hierarchical data) between databases. If character data containing row separators is exported to a delimited ASCII (DEL) file and processed by a text transfer program, fields containing the row separators will shrink or expand.

The data in ASC and DEL files is assumed to be in the code page of the client application performing the import. PC/IXF files, which allow for different code pages, are recommended when importing data in different code pages. If the PC/IXF file and the import utility are in the same code page, processing occurs as for a regular application. If the two differ, and the FORCEIN option is specified, the import utility assumes that data in the PC/IXF file has the same code page as the application performing the import. This occurs even if there is a conversion table for the two code pages. If the two differ, the FORCEIN option is not specified, and there is a conversion table, all data in the PC/IXF file will be converted from the file code page to the application code page. If the two differ, the FORCEIN option is not specified, and there is no conversion table, the import operation will fail. This applies only to PC/IXF files on DB2 for AIX clients.

For table objects on an 8KB page that are close to the limit of 1012 columns, import of PC/IXF data files may cause DB2 to return an error, because the maximum size of an SQL statement was exceeded. This situation can occur only if the columns are of type CHAR, VARCHAR, or CLOB. The restriction does not apply to import of DEL or ASC files.

DB2 Connect can be used to import data to DRDA servers such as DB2 for OS/390, DB2 for VM and VSE, and DB2 for OS/400. Only PC/IXF import (INSERT option) is supported. The restartcnt parameter, but not the commitcnt parameter, is also supported.

When using the CREATE option with typed tables, create every sub-table defined in the PC/IXF file; sub-table definitions cannot be altered. When using options other than CREATE with typed tables, the traversal order list enables one to specify the traverse order; therefore, the traversal order list must match the one used during the export operation. For the PC/IXF file format, one need only specify the target sub-table name, and use the traverse order stored in the file. The import utility can be used to recover a table previously exported to a PC/IXF file. The table returns to the state it was in when exported.

Data cannot be imported to a system table, a declared temporary table, or a summary table.

Views cannot be created through the import utility.

On the Windows operating system:
- Importing logically split PC/IXF files is not supported.
- Importing bad format PC/IXF or WSF files is not supported.

Federated considerations:

When using the db2Import API and the INSERT, UPDATE, or INSERT_UPDATE parameters, you must ensure that you have CONTROL privilege on the participating nickname. You must ensure that the nickname you wish to use when doing an import operation already exists.

Related tasks:
- "Importing data" in *Data Movement Utilities Guide and Reference*

Related reference:
- "IMPORT " on page 512
- "SQLCA data structure" in *Administrative API Reference*
- "sqldcol data structure" in *Administrative API Reference*
- "sqlu_media_list data structure" in *Administrative API Reference*
- "IMPORT command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "db2Export - Export data from a database" on page 684
- "db2Load - Load data into a table" on page 723

Related samples:
- "expsamp.sqb -- Export and import tables with table data to a DRDA database (IBM COBOL)"
- "impexp.sqb -- Export and import tables with table data (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2Inspect - Inspect database for architectural integrity

Inspects the database for architectural integrity and checks the pages of the database for page consistency.

Scope:

In a single partition database environment, the scope is the single database partition only. In a partitioned database environment it is the collection of all logical database partitions defined in db2nodes.cfg. For partitioned tables, the scope for database and table space level inspection includes individual data partitions and non-partitioned indexes. Table level inspection for a partitioned table checks all the data partitions and indexes in a table, rather than checking a single data partition or index.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint
- dbadm
- CONTROL privilege on the table

Required connection:

Database

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Inspect (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InspectStruct
{
   char *piTablespaceName;
   char *piTableName;
   char *piSchemaName;
   char *piResultsName;
   char *piDataFileName;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iAction;
   db2int32 iTablespaceID;
   db2int32 iObjectID;
   db2Uint32 iFirstPage;
   db2Uint32 iNumberOfPages;
   db2Uint32 iFormatType;
   db2Uint32 iOptions;
   db2Uint32 iBeginCheckOption;
   db2int32 iLimitErrorReported;
   db2Uint16 iObjectErrorState;
   db2Uint16 iCatalogToTablespace;
   db2Uint16 iKeepResultfile;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   db2Uint16 iLevelObjectData;
   db2Uint16 iLevelObjectIndex;
   db2Uint16 iLevelObjectLong;
   db2Uint16 iLevelObjectLOB;
   db2Uint16 iLevelObjectBlkMap;
   db2Uint16 iLevelExtentMap;
   db2Uint16 iLevelObjectXML;
} db2InspectStruct;

SQL_API_RC SQL_API_FN
  db2gInspect (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInspectStruct
{
   char *piTablespaceName;
   char *piTableName;
   char *piSchemaName;
   char *piResultsName;
   char *piDataFileName;
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint32 iResultsNameLength;
   db2Uint32 iDataFileNameLength;
   db2Uint32 iTablespaceNameLength;
   db2Uint32 iTableNameLength;
   db2Uint32 iSchemaNameLength;
   db2Uint32 iAction;
   db2int32 iTablespaceID;
   db2int32 iObjectID;
   db2Uint32 iFirstPage;
   db2Uint32 iNumberOfPages;
   db2Uint32 iFormatType;
   db2Uint32 iOptions;
   db2Uint32 iBeginCheckOption;
   db2int32 iLimitErrorReported;
   db2Uint16 iObjectErrorState;
   db2Uint16 iCatalogToTablespace;
   db2Uint16 iKeepResultfile;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   db2Uint16 iLevelObjectData;
   db2Uint16 iLevelObjectIndex;
   db2Uint16 iLevelObjectLong;
   db2Uint16 iLevelObjectLOB;
   db2Uint16 iLevelObjectBlkMap;
   db2Uint16 iLevelExtentMap;
   db2Uint16 iLevelObjectXML;
} db2gInspectStruct;
```

db2Inspect API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
: Input. A pointer to the db2InspectStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2InspectStruct data structure parameters:

piTablespaceName
: Input. A string containing the table space name. The table space must be identified for operations on a table space. If the pointer is NULL, the table space ID value is used as input.

piTableName
: Input. A string containing the table name. The table must be identified for

operations on a table or a table object. If the pointer is NULL, the table space ID and table object ID values are used as input.

piSchemaName
Input. A string containing the schema name.

piResultsName
Input. A string containing the name for results output file. This input must be provided. The file will be written out to the diagnostic data directory path.

piDataFileName
Input. Reserved for future use. Must be set to NULL.

piNodeList
Input. A pointer to an array of database partition numbers on which to perform the operation.

iAction
Input. Specifies the inspect action. Valid values (defined in the db2ApiDf header file, which is located in the include directory)are:

DB2INSPECT_ACT_CHECK_DB
Inspect the entire database.

DB2INSPECT_ACT_CHECK_TABSPACE
Inspect a table space.

DB2INSPECT_ACT_CHECK_TABLE
Inspect a table.

DB2INSPECT_ACT_FORMAT_XML
Format an XML object page.

DB2INSPECT_ACT_ROWCMPEST_TBL
Estimate row compression effectiveness on a table.

iTablespaceID
Input. Specifies the table space ID. If the table space must be identified, the table space ID value is used as input if the pointer to table space name is NULL.

iObjectID
Input. Specifies the object ID. If the table must be identified, the object ID value is used as input if the pointer to table name is NULL.

iBeginCheckOption
Input. Option for check database or check table space operation to indicate where operation should begin. It must be set to zero to begin from the normal start. Values are:

DB2INSPECT_BEGIN_TSPID
Use this value for check database to begin with the table space specified by the table space ID field, the table space ID must be set.

DB2INSPECT_BEGIN_TSPID_OBJID
Use this value for check database to begin with the table specified by the table space ID and object ID field. To use this option, the table space ID and object ID must be set.

DB2INSPECT_BEGIN_OBJID
Use this value for check table space to begin with the table specified by the object ID field, the object ID must be set.

iLimitErrorReported
Input. Specifies the reporting limit of the number of pages in error for an object. Specify the number you want to use as the limit value or specify one the following values:

DB2INSPECT_LIMIT_ERROR_DEFAULT
Use this value to specify that the maximum number of pages in error to be reported is the extent size of the object.

DB2INSPECT_LIMIT_ERROR_ALL
Use this value to report all pages in error.

iObjectErrorState
Input. Specifies whether to scan objects in error state. Valid values are:

DB2INSPECT_ERROR_STATE_NORMAL
Process object only in normal state.

DB2INSPECT_ERROR_STATE_ALL
Process all objects, including objects in error state.

iKeepResultfile
Input. Specifies result file retention. Valid values are:

DB2INSPECT_RESFILE_CLEANUP
If errors are reported, the result output file will be retained. Otherwise, the result file will be removed at the end of the operation.

DB2INSPECT_RESFILE_KEEP_ALWAYS
The result output file will be retained.

iAllNodeFlag
Input. Indicates whether the operation is to be applied to all nodes defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to all nodes in a node list that is passed in pNodeList.

DB2_ALL_NODES
Apply to all nodes. pNodeList should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all nodes except those in a node list that is passed in pNodeList.

iNumNodes
Input. Specifies the number of nodes in the pNodeList array.

iLevelObjectData
Input. Specifies processing level for data object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectIndex
Input. Specifies processing level for index object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectLong
Input. Specifies processing level for long object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectLOB
Input. Specifies processing level for LOB object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectBlkMap
Input. Specifies processing level for block map object. Valid values are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelExtentMap
Input. Specifies processing level for extent map. Valid values (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

iLevelObjectXML
Input. Specifies processing level for XML object. Valid values (defined in the db2ApiDf header file, which is located in the include directory) are:

DB2INSPECT_LEVEL_NORMAL
Level is normal.

DB2INSPECT_LEVEL_LOW
Level is low.

DB2INSPECT_LEVEL_NONE
Level is none.

db2gInspectStruct data structure specific parameters:

iResultsNameLength
Input. The string length of the results file name.

iDataFileNameLength
Input. The string length of the data output file name.

iTablespaceNameLength
Input. The string length of the table space name.

iTableNameLength
Input. The string length of the table name.

iSchemaNameLength
Input. The string length of the schema name.

Usage notes:

The online inspect processing will access database objects using isolation level uncommitted read. Commit processing will be done during the inspect processing. It is advisable to end the unit of work by committing or rolling back changes, by executing a COMMIT or ROLLBACK statement respectively, before starting the inspect operation.

The inspect check processing will write out unformatted inspection data results to the result file. The file will be written out to the diagnostic data directory path. If there are no errors found by the check processing, the result output file will be erased at the end of the inspect operation. If there are errors found by the check processing, the result output file will not be erased at the end of the inspect operation. To see the inspection details, format the inspection result output file with the db2inspf utility.

In a partitioned database environment, the extension of the result output file will correspond to the database partition number. The file is located in the database manager diagnostic data directory path.

A unique results output file name must be specified. If the result output file already exists, the operation will not be processed.

The processing of table spaces will process only the objects that reside in that table space.

Related reference:
- "SQLCA data structure" in *Administrative API Reference*
- "INSPECT " on page 525

db2InstanceQuiesce - Quiesce instance

Forces all users off the instance, immediately rolls back all active transaction, and puts the database into quiesce mode. This API provides exclusive access to the instance. During this quiesced period, system administration can be performed on the instance. After administration is complete, you can unquiesce the database using the db2DatabaseUnquiesce API. This API allows other users to connect to the database without having to shut down and perform another database start.

In this mode only groups or users with QUIESCE CONNECT authority and sysadm, sysmaint, or sysctrl authority will have access to the database and its objects.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2InstanceQuiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InsQuiesceStruct
{
             char *piInstanceName;
             char *piUserId;
             char *piGroupId;
             db2Uint32 iImmediate;
             db2Uint32 iForce;
             db2Uint32 iTimeout;
} db2InsQuiesceStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceQuiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInsQuiesceStruct
{
             db2Uint32 iInstanceNameLen;
             char *piInstanceName;
             db2Uint32 iUserIdLen;
             char *piUserId;
             db2Uint32 iGroupIdLen;
             char *piGroupId;
```

```
        db2Uint32 iImmediate;
        db2Uint32 iForce;
        db2Uint32 iTimeout;
} db2gInsQuiesceStruct;
```

db2InstanceQuiesce API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
: Input. A pointer to the db2InsQuiesceStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2InsQuiesceStruct data structure parameters:

piInstanceName
: Input. The instance name.

piUserId
: Input. The name of the a user who will be allowed access to the instance while it is quiesced.

piGroupId
: Input. The name of a group that will be allowed access to the instance while the instance is quiesced.

iImmediate
: Input. Valid values are:

  TRUE=1
  : Force the applications immediately.

  FALSE=0
  : Deferred force. Applications will wait the number of minutes specified by iTimeout parameter to let their current units of work be completed, and then will terminate. If this deferred force cannot be completed within the number of minutes specified by iTimeout parameter, the quiesce operation will fail.

iForce Input. Reserved for future use.

iTimeout
: Input. Specifies the time, in minutes, to wait for applications to commit the current unit of work. If iTimeout is not specified, in a single-partition database environment, the default value is 10 minutes. In a partitioned database environment the value specified by the start_stop_timeout database manager configuration parameter will be used.

db2gInsQuiesceStruct data structure specific parameters:

iInstanceNameLen
: Input. Specifies the length in bytes of piInstanceName.

iUserIdLen
: Input. Specifies the length in bytes of piUserID.

iGroupIdLen
: Input. Specifies the length in bytes of piGroupId.

Related reference:

- "QUIESCE " on page 567
- "db2InstanceUnquiesce - Unquiesce instance" on page 722
- "SQLCA data structure" in *Administrative API Reference*
- "db2InstanceStart - Start instance" on page 714
- "db2InstanceStop - Stop instance" on page 719

db2InstanceStart - Start instance

Starts a local or remote instance.

Scope:

In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, db2nodes.cfg.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint

Required connection:

None

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2InstanceStart (
            db2Uint32 versionNumber,
            void * pParmStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InstanceStartStruct
{
            db2int8 iIsRemote;
            char *piRemoteInstName;
            db2DasCommData * piCommData;
            db2StartOptionsStruct * piStartOpts;
} db2InstanceStartStruct;

typedef SQL_STRUCTURE db2DasCommData
{
            db2int8 iCommParam;
            char *piNodeOrHostName;
            char *piUserId;
            char *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StartOptionsStruct
{
            db2Uint32 iIsProfile;
```

```
             char *piProfile;
             db2Uint32 iIsNodeNum;
             db2NodeType iNodeNum;
             db2Uint32 iOption;
             db2Uint32 iIsHostName;
             char *piHostName;
             db2Uint32 iIsPort;
             db2PortType iPort;
             db2Uint32 iIsNetName;
             char *piNetName;
             db2Uint32 iTblspaceType;
             db2NodeType iTblspaceNode;
             db2Uint32 iIsComputer;
             char *piComputer;
             char *piUserName;
             char *piPassword;
             db2QuiesceStartStruct iQuiesceOpts;
} db2StartOptionsStruct;

typedef SQL_STRUCTURE db2QuiesceStartStruct
{
             db2int8 iIsQRequested;
             char *piQUsrName;
             char *piQGrpName;
             db2int8 iIsQUsrGrpDef;
} db2QuiesceStartStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceStart (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInstanceStStruct
{
             db2int8 iIsRemote;
             db2Uint32 iRemoteInstLen;
             char *piRemoteInstName;
             db2gDasCommData * piCommData;
             db2gStartOptionsStruct * piStartOpts;
} db2gInstanceStStruct;

typedef SQL_STRUCTURE db2gDasCommData
{
             db2int8 iCommParam;
             db2Uint32 iNodeOrHostNameLen;
             char *piNodeOrHostName;
             db2Uint32 iUserIdLen;
             char *piUserId;
             db2Uint32 iUserPwLen;
             char *piUserPw;
} db2gDasCommData;

typedef SQL_STRUCTURE db2gStartOptionsStruct
{
             db2Uint32 iIsProfile;
             char *piProfile;
             db2Uint32 iIsNodeNum;
             db2NodeType iNodeNum;
             db2Uint32 iOption;
             db2Uint32 iIsHostName;
             char *piHostName;
             db2Uint32 iIsPort;
             db2PortType iPort;
             db2Uint32 iIsNetName;
             char *piNetName;
             db2Uint32 iTblspaceType;
```

```
            db2NodeType iTblspaceNode;
            db2Uint32 iIsComputer;
            char *piComputer;
            char *piUserName;
            char *piPassword;
            db2gQuiesceStartStruct iQuiesceOpts;
} db2gStartOptionsStruct;

typedef SQL_STRUCTURE db2gQuiesceStartStruct
{
            db2int8 iIsQRequested;
            db2Uint32 iQUsrNameLen;
            char *piQUsrName;
            db2Uint32 iQGrpNameLen;
            char *piQGrpName;
            db2int8 iIsQUsrGrpDef;
} db2gQuiesceStartStruct;
```

db2InstanceStart API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2InstanceStartStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2InstanceStartStruct data structure parameters:

iIsRemote
Input. An indicator set to constant integer value TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piRemoteInstName
Input. The name of the remote instance.

piCommData
Input. A pointer to the db2DasCommData structure.

piStartOpts
Input. A pointer to the db2StartOptionsStruct structure.

db2DasCommData data structure parameters:

iCommParam
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piNodeOrHostName
Input. The database partition or hostname.

piUserId
Input. The user name.

piUserPw
Input. The user password.

db2StartOptionsStruct data structure parameters:

iIsProfile
Input. Indicates whether a profile is specified. If this field indicates that a profile is not specified, the file db2profile is used.

piProfile
Input. The name of the profile file to be executed at each node to define the DB2 environment (MPP only). This file is executed before the nodes are started. The default value is db2profile.

iIsNodeNum
Input. Indicates whether a node number is specified. If specified, the start command only affects the specified node.

iNodeNum
Input. The database partition number.

iOption
Input. Specifies an action. Valid values for OPTION (defined in sqlenv header file, located in the include directory) are: are:

SQLE_NONE
Issue the normal db2start operation.

SQLE_ADDNODE
Issue the ADD NODE command.

SQLE_RESTART
Issue the RESTART DATABASE command.

SQLE_STANDALONE
Start the node in STANDALONE mode.

iIsHostName
Input. Indicates whether a host name is specified.

piHostName
Input. The system name.

iIsPort
Input. Indicates whether a port number is specified.

iPort Input. The port number.

iIsNetName
Input. Indicates whether a net name is specified.

piNetName
Input. The network name.

iTblspaceType
Input. Specifies the type of system temporary table space definitions to be used for the node being added. Valid values are:

SQLE_TABLESPACES_NONE
Do not create any system temporary table spaces.

SQLE_TABLESPACES_LIKE_NODE
The containers for the system temporary table spaces should be the same as those for the specified node.

SQLE_TABLESPACES_LIKE_CATALOG
The containers for the system temporary table spaces should be the same as those for the catalog node of each database.

iTblspaceNode
Input. Specifies the node number from which the system temporary table space definitions should be obtained. The node number must exist in the db2nodes.cfg file, and is only used if the tblspace_type field is set to SQLE_TABLESPACES_LIKE_NODE.

iIsComputer
Input. Indicates whether a computer name is specified. Valid on the Windows operating system only.

piComputer
Input. Computer name. Valid on the Windows operating system only.

piUserName
Input. Logon account user name. Valid on the Windows operating system only.

piPassword
Input. The password corresponding to the logon account user name.

iQuiesceOpts
Input. A pointer to the db2QuiesceStartStruct structure.

db2QuiesceStartStruct data structure parameters:

iIsQRequested
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if quiesce is requested.

piQUsrName
Input. The quiesced username.

piQGrpName
Input. The quiesced group name.

iIsQUsrGrpDef
Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if a quiesced user or quiesced group is defined.

db2gInstanceStStruct data structure specific parameters:

iRemoteInstLen
Input. Specifies the length in bytes of piRemoteInstName.

db2gDasCommData data structure specific parameters:

iNodeOrHostNameLen
Input. Specifies the length in bytes of piNodeOrHostName.

iUserIdLen
Input. Specifies the length in bytes of piUserId.

iUserPwLen
Input. Specifies the length in bytes of piUserPw.

db2gQuiesceStartStruct data structure specific parameters:

iQUsrNameLen
Input. Specifies the length in bytes of piQusrName.

iQGrpNameLen
Input. Specifies the length in bytes of piQGrpName.

Related reference:
- “START DATABASE MANAGER ” on page 618
- “db2InstanceStop - Stop instance” on page 719
- “SQLCA data structure” in *Administrative API Reference*
- “db2InstanceQuiesce - Quiesce instance” on page 712

Related samples:

- "instart.c -- Stop and start the current local instance (C)"
- "instart.C -- Stop and start the current local instance (C++)"

db2InstanceStop - Stop instance

Stops the local or remote DB2 instance.

Scope:

In a single-partition database environment, the scope is that single database partition only. In a partitioned database environment, it is the collection of all logical database partition servers defined in the node configuration file, db2nodes.cfg.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

None

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2InstanceStop (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InstanceStopStruct
{
             db2int8 iIsRemote;
             char *piRemoteInstName;
             db2DasCommData * piCommData;
             db2StopOptionsStruct * piStopOpts;
} db2InstanceStopStruct;

typedef SQL_STRUCTURE db2DasCommData
{
             db2int8 iCommParam;
             char *piNodeOrHostName;
             char *piUserId;
             char *piUserPw;
} db2DasCommData;

typedef SQL_STRUCTURE db2StopOptionsStruct
{
             db2Uint32 iIsProfile;
             char *piProfile;
             db2Uint32 iIsNodeNum;
             db2NodeType iNodeNum;
             db2Uint32 iStopOption;
             db2Uint32 iCallerac;
} db2StopOptionsStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceStop (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInstanceStopStruct
{
             db2int8 iIsRemote;
             db2Uint32 iRemoteInstLen;
             char *piRemoteInstName;
             db2gDasCommData * piCommData;
             db2StopOptionsStruct * piStopOpts;
} db2gInstanceStopStruct;

typedef SQL_STRUCTURE db2gDasCommData
{
             db2int8 iCommParam;
             db2Uint32 iNodeOrHostNameLen;
             char *piNodeOrHostName;
             db2Uint32 iUserIdLen;
             char *piUserId;
             db2Uint32 iUserPwLen;
             char *piUserPw;
} db2gDasCommData;
```

db2InstanceStop API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
: Input. A pointer to the db2InstanceStopStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2InstanceStopStruct data structure parameters:

iIsRemote
: Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piRemoteInstName
: Input. The name of the remote instance.

piCommData
: Input. A pointer to the db2DasCommData structure.

piStopOpts
: Input. A pointer to the db2StopOptionsStruct structure.

db2DasCommData data structure parameters:

iCommParam
: Input. An indicator set to TRUE or FALSE. This parameter should be set to TRUE if this is a remote start.

piNodeOrHostName
: Input. The database partition or hostname.

piUserId
: Input. The user name.

piUserPw
: Input. The user password.

db2StopOptionsStruct data structure parameters:

iIsProfile
: Input. Indicates whether a profile is specified. Possible values are TRUE and FALSE. If this field indicates that a profile is not specified, the file db2profile is used.

piProfile
: Input. The name of the profile file that was executed at startup to define the DB2 environment for those nodes that were started (MPP only). If a profile for the db2InstanceStart API was specified, the same profile must be specified here.

iIsNodeNum
: Input. Indicates whether a node number is specified. Possible values are TRUE and FALSE. If specified, the stop command only affects the specified node.

iNodeNum
: Input. The database partition number.

iStopOption
: Input. Option. Valid values are:

  SQLE_NONE
  : Issue the normal db2stop operation.

  SQLE_FORCE
  : Issue the FORCE APPLICATION (ALL) command.

  SQLE_DROP
  : Drop the node from the db2nodes.cfg file.

iCallerac
: Input. This field is valid only for the SQLE_DROP value of the OPTION field. Valid values are:

  SQLE_DROP
  : Initial call. This is the default value.

  SQLE_CONTINUE
  : Subsequent call. Continue processing after a prompt.

  SQLE_TERMINATE
  : Subsequent call. Terminate processing after a prompt.

db2gInstanceStopStruct data structure specific parameters:

iRemoteInstLen
: Input. Specifies the length in bytes of piRemoteInstName.

db2gDasCommData data structure specific parameters:

iNodeOrHostNameLen
: Input. Specifies the length in bytes of piNodeOrHostName.

iUserIdLen
: Input. Specifies the length in bytes of piUserId.

iUserPwLen
Input. Specifies the length in bytes of piUserPw.

Related reference:

- "STOP DATABASE MANAGER " on page 624
- "db2InstanceStart - Start instance" on page 714
- "SQLCA data structure" in *Administrative API Reference*
- "db2InstanceQuiesce - Quiesce instance" on page 712
- "db2InstanceUnquiesce - Unquiesce instance" on page 722

Related samples:

- "instart.c -- Stop and start the current local instance (C)"
- "instart.C -- Stop and start the current local instance (C++)"

db2InstanceUnquiesce - Unquiesce instance

Unquiesce all databases in the instance.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2InstanceUnquiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2InsUnquiesceStruct
{
             char *piInstanceName;
} db2InsUnquiesceStruct;

SQL_API_RC SQL_API_FN
  db2gInstanceUnquiesce (
             db2Uint32 versionNumber,
             void * pParmStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gInsUnquiesceStruct
{
             db2Uint32 iInstanceNameLen;
             char *piInstanceName;
} db2gInsUnquiesceStruct;
```

db2InstanceUnquiesce API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
: Input. A pointer to the db2InsUnquiesceStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2InsUnquiesceStruct data structure parameters:

piInstanceName
: Input. The instance name.

db2gInsUnquiesceStruct data structure specific parameters:

iInstanceNameLen
: Input. Specifies the length in bytes of piInstanceName.

Related reference:
- "UNQUIESCE " on page 626
- "db2InstanceQuiesce - Quiesce instance" on page 712
- "SQLCA data structure" in *Administrative API Reference*
- "db2InstanceStop - Stop instance" on page 719

db2Load - Load data into a table

Loads data into a DB2 table. Data residing on the server may be in the form of a file, cursor, tape, or named pipe. Data residing on a remotely connected client may be in the form of a fully qualified file, a cursor, or named pipe. Although faster than the import utility, the load utility does not support loading data at the hierarchy level or loading into a nickname.

Authorization:

One of the following:
- sysadm
- dbadm
- load authority on the database and:
  - INSERT privilege on the table when the load utility is invoked in INSERT mode, TERMINATE mode (to terminate a previous load insert operation), or RESTART mode (to restart a previous load insert operation)
  - INSERT and DELETE privilege on the table when the load utility is invoked in REPLACE mode, TERMINATE mode (to terminate a previous load replace operation), or RESTART mode (to restart a previous load replace operation)
  - INSERT privilege on the exception table, if such a table is used as part of the load operation.

Note: In general, all load processes and all DB2 server processes are owned by the instance owner. All of these processes use the identification of the instance owner to access needed files. Therefore, the instance owner must have read access to the input files, regardless of who invokes the command.

Required connection:

Database. If implicit connect is enabled, a connection to the default database is established. Utility access to Linux, UNIX, or Windows database servers from Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment.

Instance. An explicit attachment is not required. If a connection to the database has been established, an implicit attachment to the local instance is attempted.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Load (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2LoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2PartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
} db2LoadStruct;

typedef SQL_STRUCTURE db2LoadUserExit
{
   db2Char iSourceUserExitCmd;
   struct db2Char *piInputStream;
   struct db2Char *piInputFileName;
   struct db2Char *piOutputFileName;
   db2Uint16 *piEnableParallelism;
} db2LoadUserExit;

typedef SQL_STRUCTURE db2LoadIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   char *piUseTablespace;
   db2Uint32 iSavecount;
   db2Uint32 iDataBufferSize;
   db2Uint32 iSortBufferSize;
   db2Uint32 iWarningcount;
   db2Uint16 iHoldQuiesce;
   db2Uint16 iCpuParallelism;
   db2Uint16 iDiskParallelism;
   db2Uint16 iNonrecoverable;
   db2Uint16 iIndexingMode;
   db2Uint16 iAccessLevel;
   db2Uint16 iLockWithForce;
   db2Uint16 iCheckPending;
   char iRestartphase;
   char iStatsOpt;
   db2Uint16 iSetIntegrityPending;
   struct db2LoadUserExit *piSourceUserExit;
} db2LoadIn;

typedef SQL_STRUCTURE db2LoadOut
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsLoaded;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsDeleted;
   db2Uint64 oRowsCommitted;
} db2LoadOut;

typedef SQL_STRUCTURE db2PartLoadIn
{
   char *piHostname;
   char *piFileTransferCmd;
   char *piPartFileLocation;
   struct db2LoadNodeList *piOutputNodes;
   struct db2LoadNodeList *piPartitioningNodes;
   db2Uint16 *piMode;
   db2Uint16 *piMaxNumPartAgents;
   db2Uint16 *piIsolatePartErrs;
   db2Uint16 *piStatusInterval;
   struct db2LoadPortRange *piPortRange;
   db2Uint16 *piCheckTruncation;
   char *piMapFileInput;
   char *piMapFileOutput;
   db2Uint16 *piTrace;
   db2Uint16 *piNewline;
   char *piDistfile;
   db2Uint16 *piOmitHeader;
   SQL_PDB_NODE_TYPE *piRunStatDBPartNum;
} db2PartLoadIn;

typedef SQL_STRUCTURE db2LoadNodeList
{
   SQL_PDB_NODE_TYPE *piNodeList;
   db2Uint16 iNumNodes;
} db2LoadNodeList;

typedef SQL_STRUCTURE db2LoadPortRange
{
   db2Uint16 iPortMin;
   db2Uint16 iPortMax;
} db2LoadPortRange;

typedef SQL_STRUCTURE db2PartLoadOut
{
   db2Uint64 oRowsRdPartAgents;
   db2Uint64 oRowsRejPartAgents;
   db2Uint64 oRowsPartitioned;
   struct db2LoadAgentInfo *poAgentInfoList;
   db2Uint32 iMaxAgentInfoEntries;
   db2Uint32 oNumAgentInfoEntries;
} db2PartLoadOut;

typedef SQL_STRUCTURE db2LoadAgentInfo
{
   db2int32 oSqlcode;
   db2Uint32 oTableState;
   SQL_PDB_NODE_TYPE oNodeNum;
   db2Uint16 oAgentType;
} db2LoadAgentInfo;
```

```
SQL_API_RC SQL_API_FN
  db2gLoad (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadStruct
{
   struct sqlu_media_list *piSourceList;
   struct sqlu_media_list *piLobPathList;
   struct sqldcol *piDataDescriptor;
   struct sqlchar *piActionString;
   char *piFileType;
   struct sqlchar *piFileTypeMod;
   char *piLocalMsgFileName;
   char *piTempFilesPath;
   struct sqlu_media_list *piVendorSortWorkPaths;
   struct sqlu_media_list *piCopyTargetList;
   db2int32 *piNullIndicators;
   struct db2gLoadIn *piLoadInfoIn;
   struct db2LoadOut *poLoadInfoOut;
   struct db2gPartLoadIn *piPartLoadInfoIn;
   struct db2PartLoadOut *poPartLoadInfoOut;
   db2int16 iCallerAction;
   db2Uint16 iFileTypeLen;
   db2Uint16 iLocalMsgFileLen;
   db2Uint16 iTempFilesPathLen;
} db2gLoadStruct;

typedef SQL_STRUCTURE db2gLoadIn
{
   db2Uint64 iRowcount;
   db2Uint64 iRestartcount;
   char *piUseTablespace;
   db2Uint32 iSavecount;
   db2Uint32 iDataBufferSize;
   db2Uint32 iSortBufferSize;
   db2Uint32 iWarningcount;
   db2Uint16 iHoldQuiesce;
   db2Uint16 iCpuParallelism;
   db2Uint16 iDiskParallelism;
   db2Uint16 iNonrecoverable;
   db2Uint16 iIndexingMode;
   db2Uint16 iAccessLevel;
   db2Uint16 iLockWithForce;
   db2Uint16 iCheckPending;
   char iRestartphase;
   char iStatsOpt;
   db2Uint16 iUseTablespaceLen;
   db2Uint16 iSetIntegrityPending;
   struct db2LoadUserExit *piSourceUserExit;
} db2gLoadIn;

typedef SQL_STRUCTURE db2gPartLoadIn
{
   char *piHostname;
   char *piFileTransferCmd;
   char *piPartFileLocation;
   struct db2LoadNodeList *piOutputNodes;
   struct db2LoadNodeList *piPartitioningNodes;
   db2Uint16 *piMode;
   db2Uint16 *piMaxNumPartAgents;
   db2Uint16 *piIsolatePartErrs;
   db2Uint16 *piStatusInterval;
   struct db2LoadPortRange *piPortRange;
   db2Uint16 *piCheckTruncation;
   char *piMapFileInput;
   char *piMapFileOutput;
   db2Uint16 *piTrace;
   db2Uint16 *piNewline;
   char *piDistfile;
   db2Uint16 *piOmitHeader;
   void *piReserved1;
   db2Uint16 iHostnameLen;
   db2Uint16 iFileTransferLen;
   db2Uint16 iPartFileLocLen;
   db2Uint16 iMapFileInputLen;
   db2Uint16 iMapFileOutputLen;
   db2Uint16 iDistfileLen;
} db2gPartLoadIn;
```

db2Load API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2LoadStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2LoadStruct data structure parameters:

piSourceList
Input. A pointer to an sqlu_media_list structure used to provide a list of source files, devices, vendors, pipes, or SQL statements.

The information provided in this structure depends on the value of the media_type field. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_SQL_STMT
If the media_type field is set to this value, the caller provides an SQL query through the pStatement field of the target field. The pStatement field is of type sqlu_statement_entry. The sessions field must be set to the value of 1, since the load utility only accepts a single SQL query per load.

SQLU_SERVER_LOCATION
If the media_type field is set to this value, the caller provides information through sqlu_location_entry structures. The sessions field indicates the number of sqlu_location_entry structures provided. This is used for files, devices, and named pipes.

SQLU_CLIENT_LOCATION
If the media_type field is set to this value, the caller provides information through sqlu_location_entry structures. The sessions field indicates the number of sqlu_location_entry structures provided. This is used for fully qualified files and named pipes. Note that this media_type is only valid if the API is being called via a remotely connected client.

SQLU_TSM_MEDIA
If the media_type field is set to this value, the sqlu_vendor structure is used, where filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the

number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_OTHER_MEDIA
If the media_type field is set to this value, the sqlu_vendor structure is used, where shr_lib is the shared library name, and filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

piLobPathList
Input. A pointer to an sqlu_media_list structure. For IXF, ASC, and DEL file types, a list of fully qualified paths or devices to identify the location of the individual LOB files to be loaded. The file names are found in the IXF, ASC, or DEL files, and are appended to the paths provided.

The information provided in this structure depends on the value of the media_type field. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_LOCAL_MEDIA
If set to this value, the caller provides information through sqlu_media_entry structures. The sessions field indicates the number of sqlu_media_entry structures provided.

SQLU_TSM_MEDIA
If set to this value, the sqlu_vendor structure is used, where filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of TSM sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

SQLU_OTHER_MEDIA
If set to this value, the sqlu_vendor structure is used, where shr_lib is the shared library name, and filename is the unique identifier for the data to be loaded. There should only be one sqlu_vendor entry, regardless of the value of sessions. The sessions field indicates the number of other vendor sessions to initiate. The load utility will start the sessions with different sequence numbers, but with the same data in the one sqlu_vendor entry.

piDataDescriptor
Input. Pointer to an sqldcol structure containing information about the columns being selected for loading from the external file.

If the pFileType parameter is set to SQL_ASC, the dcolmeth field of this structure must either be set to SQL_METH_L or be set to SQL_METH_D and specifies a file name with POSITIONSFILE pFileTypeMod modifier which contains starting and ending pairs and null indicator positions. The user specifies the start and end locations for each column to be loaded.

If the file type is SQL_DEL, dcolmeth can be either SQL_METH_P or SQL_METH_D. If it is SQL_METH_P, the user must provide the source column position. If it is SQL_METH_D, the first column in the file is loaded into the first column of the table, and so on.

If the file type is SQL_IXF, dcolmeth can be one of SQL_METH_P, SQL_METH_D, or SQL_METH_N. The rules for DEL files apply here, except that SQL_METH_N indicates that file column names are to be provided in the sqldcol structure.

piActionString
Input. Pointer to an sqlchar structure, followed by an array of characters specifying an action that affects the table.

The character array is of the form:

```
"INSERT|REPLACE|RESTART|TERMINATE
INTO tbname [(column_list)]
[DATALINK SPECIFICATION datalink-spec]
[FOR EXCEPTION e_tbname]"
```

INSERT
Adds the loaded data to the table without changing the existing table data.

REPLACE
Deletes all existing data from the table, and inserts the loaded data. The table definition and the index definitions are not changed.

RESTART
Restarts a previously interrupted load operation. The load operation will automatically continue from the last consistency point in the load, build, or delete phase.

TERMINATE
Terminates a previously interrupted load operation, and rolls back the operation to the point in time at which it started, even if consistency points were passed. The states of any table spaces involved in the operation return to normal, and all table objects are made consistent (index objects may be marked as invalid, in which case index rebuild will automatically take place at next access). If the table spaces in which the table resides are not in load pending state, this option does not affect the state of the table spaces.

The load terminate option will not remove a backup pending state from table spaces.

tbname
The name of the table into which the data is to be loaded. The table cannot be a system table or a declared temporary table. An alias, or the fully qualified or unqualified table name can be specified. A qualified table name is in the form schema.tablename. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

(column_list)
A list of table column names into which the data is to be inserted. The column names must be separated by commas. If a name contains spaces or lowercase characters, it must be enclosed by quotation marks.

DATALINK SPECIFICATION datalink-spec
Specifies parameters pertaining to DB2 Data Links. These parameters can be specified using the same syntax as in the LOAD command.

FOR EXCEPTION e_tbname
: Specifies the exception table into which rows in error will be copied. The exception table is used to store copies of rows that violate unique index rules, range constraints and security policies.

piFileType
: Input. A string that indicates the format of the input data source. Supported external formats (defined in sqlutil) are:

  SQL_ASC
  : Non-delimited ASCII.

  SQL_DEL
  : Delimited ASCII, for exchange with dBase, BASIC, and the IBM Personal Decision Series programs, and many other database managers and file managers.

  SQL_IXF
  : PC version of the Integrated Exchange Format, the preferred method for exporting data from a table so that it can be loaded later into the same table or into another database manager table.

  SQL_CURSOR
  : An SQL query. The sqlu_media_list structure passed in through the piSourceList parameter is of type SQLU_SQL_STMT, and refers to an actual SQL query and not a cursor declared against one.

piFileTypeMod
: Input. A pointer to the sqlchar structure, followed by an array of characters that specify one or more processing options. If this pointer is NULL, or the structure pointed to has zero characters, this action is interpreted as selection of a default specification.

  Not all options can be used with all of the supported file types. See related link ″File type modifiers for the load utility.″

piLocalMsgFileName
: Input. A string containing the name of a local file to which output messages are to be written.

piTempFilesPath
: Input. A string containing the path name to be used on the server for temporary files. Temporary files are created to store messages, consistency points, and delete phase information.

piVendorSortWorkPaths
: Input. A pointer to the sqlu_media_list structure which specifies the Vendor Sort work directories.

piCopyTargetList
: Input. A pointer to an sqlu_media_list structure used (if a copy image is to be created) to provide a list of target paths, devices, or a shared library to which the copy image is to be written.

  The values provided in this structure depend on the value of the media_type field. Valid values for this parameter (defined in sqlutil header file, located in the include directory) are:

  SQLU_LOCAL_MEDIA
  : If the copy is to be written to local media, set the media_type to this value and provide information about the targets in

sqlu_media_entry structures. The sessions field specifies the number of sqlu_media_entry structures provided.

SQLU_TSM_MEDIA
If the copy is to be written to TSM, use this value. No further information is required.

SQLU_OTHER_MEDIA
If a vendor product is to be used, use this value and provide further information via an sqlu_vendor structure. Set the shr_lib field of this structure to the shared library name of the vendor product. Provide only one sqlu_vendor entry, regardless of the value of sessions. The sessions field specifies the number of sqlu_media_entry structures provided. The load utility will start the sessions with different sequence numbers, but with the same data provided in the one sqlu_vendor entry.

piNullIndicators
Input. For ASC files only. An array of integers that indicate whether or not the column data is nullable. There is a one-to-one ordered correspondence between the elements of this array and the columns being loaded from the data file. That is, the number of elements must equal the dcolnum field of the pDataDescriptor parameter. Each element of the array contains a number identifying a location in the data file that is to be used as a NULL indicator field, or a zero indicating that the table column is not nullable. If the element is not zero, the identified location in the data file must contain a Y or an N. A Y indicates that the table column data is NULL, and N indicates that the table column data is not NULL.

piLoadInfoIn
Input. A pointer to the db2LoadIn structure.

poLoadInfoOut
Input. A pointer to the db2LoadOut structure.

piPartLoadInfoIn
Input. A pointer to the db2PartLoadIn structure.

poPartLoadInfoOut
Output. A pointer to the db2PartLoadOut structure.

iCallerAction
Input. An action requested by the caller. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INITIAL
Initial call. This value (or SQLU_NOINTERRUPT) must be used on the first call to the API.

SQLU_NOINTERRUPT
Initial call. Do not suspend processing. This value (or SQLU_INITIAL) must be used on the first call to the API.

If the initial call or any subsequent call returns and requires the calling application to perform some action prior to completing the requested load operation, the caller action must be set to one of the following:

SQLU_CONTINUE
Continue processing. This value can only be used on subsequent calls to the API, after the initial call has returned with the utility requesting user input (for example, to respond to an end of tape

condition). It specifies that the user action requested by the utility has completed, and the utility can continue processing the initial request.

SQLU_TERMINATE
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_ABORT
Terminate processing. Causes the load utility to exit prematurely, leaving the table spaces being loaded in LOAD_PENDING state. This option should be specified if further processing of the data is not to be done.

SQLU_RESTART
Restart processing.

SQLU_DEVICE_TERMINATE
Terminate a single device. This option should be specified if the utility is to stop reading data from the device, but further processing of the data is to be done.

db2LoadUserExit data structure parameters:

iSourceUserExitCmd
Input. The fully qualified name of an executable that will be used to feed data to the utility. For security reasons, the executable must be placed within the `sqllib/bin` directory on the server. This parameter is mandatory if the piSourceUserExit structure is not NULL.

The piInputStream, piInputFileName, piOutputFileName and piEnableParallelism fields are optional. See the Data Movement Utilities Guide for a detailed description.

piInputStream
Input. A generic byte-stream that will be passed directly to the user-exit application via STDIN. You have complete control over what data is contained in this byte-stream and in what format. The load utility will simply carry this byte-stream over to the server and pass it into the user-exit application by feeding the process' STDIN (it will not codepage convert or modify the byte-stream). Your user-exit application would read the arguments from STDIN and use the data however intended.

One important attribute of this feature is the ability to hide sensitive information (such as userid/passwords). See the Data Movement Utilities Guide for a detailed description.

piInputFileName
Input. Contains the name of a fully qualified client-side file, whose contents will be passed into the user-exit application by feeding the process' STDIN.

piOutputFileName
Input. The fully qualified name of a server-side file. The STDOUT and STDERR streams of the process which is executing the user-exit application will be streamed into this file. When piEnableParallelism is RUE, multiple files will be created (one per user-exit instance), and each file name will be appended with a 3 digit numeric node-number value, such as `<filename>.000`).

piEnableParallelism
: Input. A flag indicating that the utility should attempt to parallelize the invocation of the user-exit application. See the Data Movement Utilities Guide for a detailed description.

db2LoadIn data structure parameters:

iRowcount
: Input. The number of physical records to be loaded. Allows a user to load only the first rowcnt rows in a file.

iRestartcount
: Input. Reserved for future use.

piUseTablespace
: Input. If the indexes are being rebuilt, a shadow copy of the index is built in tablespace iUseTablespaceName and copied over to the original tablespace at the end of the load. Only system temporary table spaces can be used with this option. If not specified then the shadow index will be created in the same tablespace as the index object.

  If the shadow copy is created in the same tablespace as the index object, the copy of the shadow index object over the old index object is instantaneous. If the shadow copy is in a different tablespace from the index object a physical copy is performed. This could involve considerable I/O and time. The copy happens while the table is offline at the end of a load.

  This field is ignored if iAccessLevel is SQLU_ALLOW_NO_ACCESS.

  This option is ignored if the user does not specify INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT. This option will also be ignored if INDEXING MODE AUTOSELECT is chosen and load chooses to incrementally update the index.

iSavecount
: The number of records to load before establishing a consistency point. This value is converted to a page count, and rounded up to intervals of the extent size. Since a message is issued at each consistency point, this option should be selected if the load operation will be monitored using db2LoadQuery - Load Query. If the value of savecnt is not sufficiently high, the synchronization of activities performed at each consistency point will impact performance.

  The default value is 0, meaning that no consistency points will be established, unless necessary.

iDataBufferSize
: The number of 4KB pages (regardless of the degree of parallelism) to use as buffered space for transferring data within the utility. If the value specified is less than the algorithmic minimum, the required minimum is used, and no warning is returned.

  This memory is allocated directly from the utility heap, whose size can be modified through the util_heap_sz database configuration parameter.

  If a value is not specified, an intelligent default is calculated by the utility at run time. The default is based on a percentage of the free space available in the utility heap at the instantiation time of the loader, as well as some characteristics of the table.

iSortBufferSize
Input. This option specifies a value that overrides the SORTHEAP database configuration parameter during a load operation. It is relevant only when loading tables with indexes and only when the iIndexingMode parameter is not specified as SQLU_INX_DEFERRED. The value that is specified cannot exceed the value of SORTHEAP. This parameter is useful for throttling the sort memory used by LOAD without changing the value of SORTHEAP, which would also affect general query processing.

iWarningcount
Input. Stops the load operation after warningcnt warnings. Set this parameter if no warnings are expected, but verification that the correct file and table are being used is desired. If the load file or the target table is specified incorrectly, the load utility will generate a warning for each row that it attempts to load, which will cause the load to fail. If warningcnt is 0, or this option is not specified, the load operation will continue regardless of the number of warnings issued.

If the load operation is stopped because the threshold of warnings was exceeded, another load operation can be started in RESTART mode. The load operation will automatically continue from the last consistency point. Alternatively, another load operation can be initiated in REPLACE mode, starting at the beginning of the input file.

iHoldQuiesce
Input. A flag whose value is set to TRUE if the utility is to leave the table in quiesced exclusive state after the load, and to FALSE if it is not.

iCpuParallelism
Input. The number of processes or threads that the load utility will spawn for parsing, converting and formatting records when building table objects. This parameter is designed to exploit intra-partition parallelism. It is particularly useful when loading presorted data, because record order in the source data is preserved. If the value of this parameter is zero, the load utility uses an intelligent default value at run time. Note: If this parameter is used with tables containing either LOB or LONG VARCHAR fields, its value becomes one, regardless of the number of system CPUs, or the value specified by the user.

iDiskParallelism
Input. The number of processes or threads that the load utility will spawn for writing data to the table space containers. If a value is not specified, the utility selects an intelligent default based on the number of table space containers and the characteristics of the table.

iNonrecoverable
Input. Set to SQLU_NON_RECOVERABLE_LOAD if the load transaction is to be marked as non-recoverable, and it will not be possible to recover it by a subsequent roll forward action. The rollforward utility will skip the transaction, and will mark the table into which data was being loaded as ″invalid″. The utility will also ignore any subsequent transactions against that table. After the roll forward is completed, such a table can only be dropped. With this option, table spaces are not put in backup pending state following the load operation, and a copy of the loaded data does not have to be made during the load operation. Set to SQLU_RECOVERABLE_LOAD if the load transaction is to be marked as recoverable.

iIndexingMode
Input. Specifies the indexing mode. Valid values (defined in sqlutil header file, located in the include directory) are:

SQLU_INX_AUTOSELECT
LOAD chooses between REBUILD and INCREMENTAL indexing modes.

SQLU_INX_REBUILD
Rebuild table indexes.

SQLU_INX_INCREMENTAL
Extend existing indexes.

SQLU_INX_DEFERRED
Do not update table indexes.

iAccessLevel
Input. Specifies the access level. Valid values are:

SQLU_ALLOW_NO_ACCESS
Specifies that the load locks the table exclusively.

SQLU_ALLOW_READ_ACCESS
Specifies that the original data in the table (the non-delta portion) should still be visible to readers while the load is in progress. This option is only valid for load appends, such as a load insert, and will be ignored for load replace.

iLockWithForce
Input. A boolean flag. If set to TRUE load will force other applications as necessary to ensure that it obtains table locks immediately. This option requires the same authority as the FORCE APPLICATIONS command (SYSADM or SYSCTRL).

SQLU_ALLOW_NO_ACCESS loads may force conflicting applications at the start of the load operation. At the start of the load the utility may force applications that are attempting to either query or modify the table.

SQLU_ALLOW_READ_ACCESS loads may force conflicting applications at the start or end of the load operation. At the start of the load the load utility may force applications that are attempting to modify the table. At the end of the load the load utility may force applications that are attempting to either query or modify the table.

iCheckPending
This parameter is being deprecated as of Version 9.1. Use the iSetIntegrityPending parameter instead.

iRestartphase
Input. Reserved. Valid value is a single space character ' '.

iStatsOpt
Input. Granularity of statistics to collect. Valid values are:

SQLU_STATS_NONE
No statistics to be gathered.

SQLU_STATS_USE_PROFILE
Statistics are collected based on the profile defined for the current table. This profile must be created using the RUNSTATS command. If no profile exists for the current table, a warning is returned and no statistics are collected.

iSetIntegrityPending
Input. Specifies to put the table into set integrity pending state. If the value SQLU_SI_PENDING_CASCADE_IMMEDIATE is specified, set integrity pending state will be immediately cascaded to all dependent and descendent tables. If the value SQLU_SI_PENDING_CASCADE_DEFERRED is specified, the cascade of set integrity pending state to dependent tables will be deferred until the target table is checked for integrity violations. SQLU_SI_PENDING_CASCADE_DEFERRED is the default value if the option is not specified.

db2LoadOut data structure parameters:

oRowsRead
Output. Number of records read during the load operation.

oRowsSkipped
Output. Number of records skipped before the load operation begins.

oRowsLoaded
Output. Number of rows loaded into the target table.

oRowsRejected
Output. Number of records that could not be loaded.

oRowsDeleted
Output. Number of duplicate rows deleted.

oRowsCommitted
Output. The total number of processed records: the number of records loaded successfully and committed to the database, plus the number of skipped and rejected records.

db2PartLoadIn data structure parameters:

piHostname
Input. The hostname for the iFileTransferCmd parameter. If NULL, the hostname will default to ″nohost″.

piFileTransferCmd
Input. File transfer command parameter. If not required, it must be set to NULL. See the Data Movement Guide for a full description of this parameter.

piPartFileLocation
Input. In PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the partitioned files. This location must exist on each database partition specified by the piOutputNodes option.

For the SQL_CURSOR file type, this parameter cannot be NULL and the location does not refer to a path, but to a fully qualified file name. This will be the fully qualified base file name of the partitioned files that are created on each output database partition for PARTITION_ONLY mode, or the location of the files to be read from each database partition for LOAD_ONLY mode. For PARTITION_ONLY mode, multiple files may be created with the specified base name if there are LOB columns in the target table. For file types other than SQL_CURSOR, if the value of this parameter is NULL, it will default to the current directory.

piOutputNodes
Input. The list of Load output database partitions. A NULL indicates that all nodes on which the target table is defined.

piPartitioningNodes
Input. The list of partitioning nodes. A NULL indicates the default. Refer to the Load command in the Data Movement Guide and Reference for a description of how the default is determined.

piMode
Input. Specifies the load mode for partitioned databases. Valid values (defined in db2ApiDf header file, located in the include directory) are:

- DB2LOAD_PARTITION_AND_LOAD
Data is distributed (perhaps in parallel) and loaded simultaneously on the corresponding database partitions.

- DB2LOAD_PARTITION_ONLY
Data is distributed (perhaps in parallel) and the output is written to files in a specified location on each loading database partition. For file types other than SQL_CURSOR, the name of the output file on each database partition will have the form filename.xxx, where filename is the name of the first input file specified by piSourceList and xxx is the database partition number.For the SQL_CURSOR file type, the name of the output file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY
Data is assumed to be already distributed; the distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. For file types other than SQL_CURSOR, the input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number. For the SQL_CURSOR file type, the name of the input file on each database partition will be determined by the piPartFileLocation parameter. Refer to the piPartFileLocation parameter for information about how to specify the location of the database partition file on each database partition.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_LOAD_ONLY_VERIFY_PART
Data is assumed to be already distributed, but the data file does not contain a database partition header. The distribution process is skipped, and the data is loaded simultaneously on the corresponding database partitions. During the load operation, each row is checked to verify that it is on the correct database partition. Rows containing database partition violations are placed in a dumpfile if the dumpfile file type modifier is specified. Otherwise, the rows are discarded. If database partition violations exist on a particular loading database partition, a single warning will be

written to the load message file for that database partition. The input file name on each database partition is expected to be of the form filename.xxx, where filename is the name of the first file specified by piSourceList and xxx is the 13-digit database partition number.

Note: This mode cannot be used when loading a data file located on a remote client, nor can it be used for a CLI LOAD.

DB2LOAD_ANALYZE
An optimal distribution map with even distribution across all database partitions is generated.

piMaxNumPartAgents
Input. The maximum number of partitioning agents. A NULL value indicates the default, which is 25.

piIsolatePartErrs
Input. Indicates how the load operation will react to errors that occur on individual database partitions. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOAD_SETUP_ERRS_ONLY
In this mode, errors that occur on a database partition during setup, such as problems accessing a database partition or problems accessing a table space or table on a database partition, will cause the load operation to stop on the failing database partitions but to continue on the remaining database partitions. Errors that occur on a database partition while data is being loaded will cause the entire operation to fail and rollback to the last point of consistency on each database partition.

DB2LOAD_LOAD_ERRS_ONLY
In this mode, errors that occur on a database partition during setup will cause the entire load operation to fail. When an error occurs while data is being loaded, the database partitions with errors will be rolled back to their last point of consistency. The load operation will continue on the remaining database partitions until a failure occurs or until all the data is loaded. On the database partitions where all of the data was loaded, the data will not be visible following the load operation. Because of the errors in the other database partitions the transaction will be aborted. Data on all of the database partitions will remain invisible until a load restart operation is performed. This will make the newly loaded data visible on the database partitions where the load operation completed and resume the load operation on database partitions that experienced an error.

Note: This mode cannot be used when iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified.

DB2LOAD_SETUP_AND_LOAD_ERRS
In this mode, database partition-level errors during setup or loading data cause processing to stop only on the affected database partitions. As with the DB2LOAD_LOAD_ERRS_ONLY mode, when database partition errors do occur while data is being loaded, the data on all database partitions will remain invisible until a load restart operation is performed.

Note: This mode cannot 1be used when iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified.

DB2LOAD_NO_ISOLATION
Any error during the Load operation causes the transaction to be aborted. If this parameter is NULL, it will default to DB2LOAD_LOAD_ERRS_ONLY, unless iAccessLevel is set to SQLU_ALLOW_READ_ACCESS and a copy target is also specified, in which case the default is DB2LOAD_NO_ISOLATION.

piStatusInterval
Input. Specifies the number of megabytes (MB) of data to load before generating a progress message. Valid values are whole numbers in the range of 1 to 4000. If NULL is specified, a default value of 100 will be used.

piPortRange
Input. The TCP port range for internal communication. If NULL, the port range used will be 6000-6063.

piCheckTruncation
Input. Causes Load to check for record truncation at Input/Output. Valid values are TRUE and FALSE. If NULL, the default is FALSE.

piMapFileInput
Input. Distribution map input filename. If the mode is not ANALYZE, this parameter should be set to NULL. If the mode is ANALYZE, this parameter must be specified.

piMapFileOutput
Input. Distribution map output filename. The rules for piMapFileInput apply here as well.

piTrace
Input. Specifies the number of records to trace when you need to review a dump of all the data conversion process and the output of hashing values. If NULL, the number of records defaults to 0.

piNewline
Input. Forces Load to check for newline characters at end of ASC data records if RECLEN file type modifier is also specified. Possible values are TRUE and FALSE. If NULL, the value defaults to FALSE.

piDistfile
Input. Name of the database partition distribution file. If a NULL is specified, the value defaults to ″DISTFILE″.

piOmitHeader
Input. Indicates that a distribution map header should not be included in the database partition file when using DB2LOAD_PARTITION_ONLY mode. Possible values are TRUE and FALSE. If NULL, the default is FALSE.

piRunStatDBPartNum
Specifies the database partition on which to collect statistics. The default value is the first database partition in the output database partition list.

db2LoadNodeList data structure parameters:

piNodeList
Input. An array of node numbers.

iNumNodes
Input. The number of nodes in the piNodeList array. A 0 indicates the default, which is all nodes on which the target table is defined.

db2LoadPortRange data structure parameters:

iPortMin
Input. Lower port number.

iPortMax
Input. Higher port number.

db2PartLoadOut data structure parameters:

oRowsRdPartAgents
Output. Total number of rows read by all partitioning agents.

oRowsRejPartAgents
Output. Total number of rows rejected by all partitioning agents.

oRowsPartitioned
Output. Total number of rows partitioned by all partitioning agents.

poAgentInfoList
Output. During a load operation into a partitioned database, the following load processing entities may be involved: load agents, partitioning agents, pre-partitioning agents, file transfer command agents and load-to-file agents (these are described in the Data Movement Guide). The purpose of the poAgentInfoList output parameter is to return to the caller information about each load agent that participated in a load operation. Each entry in the list contains the following information:

oAgentType
A tag indicating what kind of load agent the entry describes.

oNodeNum
The number of the database partition on which the agent executed.

oSqlcode
The final sqlcode resulting from the agent's processing.

oTableState
The final status of the table on the database partition on which the agent executed (relevant only for load agents).

It is up to the caller of the API to allocate memory for this list prior to calling the API. The caller should also indicate the number of entries for which they allocated memory in the iMaxAgentInfoEntries parameter. If the caller sets poAgentInfoList to NULL or sets iMaxAgentInfoEntries to 0, then no information will be returned about the load agents.

iMaxAgentInfoEntries
Input. The maximum number of agent information entries allocated by the user for poAgentInfoList. In general, setting this parameter to 3 times the number of database partitions involved in the load operation should be sufficient.

oNumAgentInfoEntries
Output. The actual number of agent information entries produced by the load operation. This number of entries will be returned to the user in the poAgentInfoList parameter as long as iMaxAgentInfoEntries is greater than or equal to oNumAgentInfoEntries. If iMaxAgentInfoEntries is less than

oNumAgentInfoEntries, then the number of entries returned in poAgentInfoList is equal to iMaxAgentInfoEntries.

db2LoadAgentInfo data structure parameters:

oSqlcode
Output. The final sqlcode resulting from the agent's processing.

oTableState
Output. The purpose of this output parameter is not to report every possible state of the table after the load operation. Rather, its purpose is to report only a small subset of possible tablestates in order to give the caller a general idea of what happened to the table during load processing. This value is relevant for load agents only. The possible values are:

DB2LOADQUERY_NORMAL
Indicates that the load completed successfully on the database partition and the table was taken out of the LOAD IN PROGRESS (or LOAD PENDING) state. In this case, the table still could be in SET INTEGRITY PENDING state due to the need for further constraints processing, but this will not reported as this is normal.

DB2LOADQUERY_UNCHANGED
Indicates that the load job aborted processing due to an error but did not yet change the state of the table on the database partition from whatever state it was in prior to calling db2Load. It is not necessary to perform a load restart or terminate operation on such database partitions.

DB2LOADQUERY_LOADPENDING
Indicates that the load job aborted during processing but left the table on the database partition in the LOAD PENDING state, indicating that the load job on that database partition must be either terminated or restarted.

oNodeNum
Output. The number of the database partition on which the agent executed.

oAgentType
Output. The agent type. Valid values (defined in db2ApiDf header file, located in the include directory) are :

- DB2LOAD_LOAD_AGENT
- DB2LOAD_PARTITIONING_AGENT
- DB2LOAD_PRE_PARTITIONING_AGENT
- DB2LOAD_FILE_TRANSFER_AGENT
- DB2LOAD_LOAD_TO_FILE_AGENT

db2gLoadStruct data structure specific parameters:

iFileTypeLen
Input. Specifies the length in bytes of iFileType parameter.

iLocalMsgFileLen
Input. Specifies the length in bytes of iLocalMsgFileName parameter.

iTempFilesPathLen
Input. Specifies the length in bytes of iTempFilesPath parameter.

db2gLoadIn data structure specific parameters:

iUseTablespaceLen
Input. The length in bytes of piUseTablespace parameter.

db2gPartLoadIn data structure specific parameters:

piReserved1
Reserved for future use.

iHostnameLen
Input. The length in bytes of piHostname parameter.

iFileTransferLen
Input. The length in bytes of piFileTransferCmd parameter.

iPartFileLocLen
Input. The length in bytes of piPartFileLocation parameter.

iMapFileInputLen
Input. The length in bytes of piMapFileInput parameter.

iMapFileOutputLen
Input. The length in bytes of piMapFileOutput parameter.

iDistfileLen
Input. The length in bytes of piDistfile parameter.

Usage notes:

Data is loaded in the sequence that appears in the input file. If a particular sequence is desired, the data should be sorted before a load is attempted.

The load utility builds indexes based on existing definitions. The exception tables are used to handle duplicates on unique keys. The utility does not enforce referential integrity, perform constraints checking, or update summary tables that are dependent on the tables being loaded. Tables that include referential or check constraints are placed in set integrity pending state. Summary tables that are defined with REFRESH IMMEDIATE, and that are dependent on tables being loaded, are also placed in set integrity pending state. Issue the SET INTEGRITY statement to take the tables out of set integrity pending state. Load operations cannot be carried out on replicated summary tables.

For clustering indexes, the data should be sorted on the clustering index prior to loading. The data need not be sorted when loading into an multi-dimensionally clustered (MDC) table.

Related tasks:
- "Loading data" in *Data Movement Utilities Guide and Reference*

Related reference:
- "LOAD " on page 542
- "sqldcol data structure" in *Administrative API Reference*
- "sqlu_media_list data structure" in *Administrative API Reference*
- "db2LoadQuery API - Get the status of a load operation" in *Administrative API Reference*
- "db2Export - Export data from a database" on page 684
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 692

Related samples:

- "dtformat.sqc -- Load and import data format extensions (C)"
- "tbload.sqc -- How to load into a partitioned database (C)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

db2Recover - Restore and roll forward a database

Restores and rolls forward a database to a particular point in time or to the end of the logs.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. If no database partition servers are specified, it affects all database partition servers that are listed in the db2nodes.cfg file. If a point in time is specified, the API affects all database partitions.

Authorization:

To recover an existing database, one of the following:
- sysadm
- sysctrl
- sysmaint

To recover to a new database, one of the following:
- sysadm
- sysctrl

Required connection:

To recover an existing database, a database connection is required. This API automatically establishes a connection to the specified database and will release the connection when the recover operation finishes. Instance and database, to recover to a new database. The instance attachment is required to create the database.

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Recover (
       db2Uint32 versionNumber,
       void * pDB2RecovStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RecoverStruct
{
  char *piSourceDBAlias;
  char *piUsername;
  char *piPassword;
  db2Uint32 iRecoverCallerAction;
  db2Uint32 iOptions;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  char *piStopTime;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piHistoryFile;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2RecoverStruct;

SQL_STRUCTURE sqlu_histFile
{
   SQL_PDB_NODE_TYPE nodeNum;
   unsigned short  filenameLen;
   char filename[SQL_FILENAME_SZ+1];
};

SQL_API_RC SQL_API_FN
  db2gRecover (
       db2Uint32 versionNumber,
       void * pDB2gRecoverStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRecoverStruct
{
  char *piSourceDBAlias;
  db2Uint32 iSourceDBAliasLen;
  char *piUserName;
  db2Uint32 iUserNameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  db2Uint32 iRecoverCallerAction;
  db2Uint32 iOptions;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  char *piStopTime;
  db2Uint32 iStopTimeLen;
  char *piOverflowLogPath;
  db2Uint32 iOverflowLogPathLen;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piHistoryFile;
  db2Uint32 iHistoryFileLen;
  db2Uint32 iNumChngHistoryFile;
  struct sqlu_histFile *piChngHistoryFile;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
} db2gRecoverStruct;
```

db2Recover API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2RecoverStruct.

pDB2RecoverStruct
Input. A pointer to the db2RecoverStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RecoverStruct data structure parameters:

piSourceDBAlias
Input. A string containing the database alias of the database to be recovered.

piUserName
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

iRecoverCallerAction
Input. Valid values are:

DB2RECOVER
Starts the recover operation. Specifies that the recover will run unattended, and that scenarios that normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the recover have been mounted, and utility prompts are not desired.

DB2RECOVER_RESTART
Allows the user to ignore a prior recover and start over from the beginning.

DB2RECOVER_CONTINUE
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2RECOVER_LOADREC_TERM
Terminate all devices being used by load recovery.

DB2RECOVER_DEVICE_TERM
Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2RECOVER_PARM_CHK_ONLY
Used to validate parameters without performing a recover operation. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.

DB2RECOVER_DEVICE_TERMINATE
Removes a particular device from the list of devices used by the recover operation. When a particular device has exhausted its input, recover will return a warning to the caller. Call the recover utility again with this caller action to remove the device that generated the warning from the list of devices being used.

iOptions
Input. Valid values are:

- DB2RECOVER_EMPTY_FLAG
No flags specified.

- DB2RECOVER_LOCAL_TIME
Indicates that the value specified for the stop time by piStopTime is in local time, not GMT. This is the default setting.

- DB2RECOVER_GMT_TIME
This flag indicates that the value specified for the stop time by piStopTime is in GMT (Greenwich Mean Time).

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

piStopTime
Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_) caller actions.

piOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the location specified by the logpath configuration parameter before they can be used by this utility. This can be a problem if the user does not have sufficient space in the log path. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the log path, and then in the overflow log path. The log files needed for table space rollforward recovery can be brought into either the log path or the overflow log path. If the caller does not specify an overflow log path, the default value is the log path.

In a partitioned database environment, the overflow log path must be a valid, fully qualified path; the default path is the default overflow log path for each database partition. In a single-partition database environment, the overflow log path can be relative if the server is local.

iNumChngLgOvrflw
Input. Partitioned database environments only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

piChngLogOvrflw
Input. Partitioned database environments only. A pointer to a structure containing the fully qualified names of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES
Apply to all database partition servers. piNodeList should be NULL. This is the default value.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY
Apply to the catalog partition only. piNodeList should be NULL.

iNumNodes
Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the rollforward recovery.

iNumNodeInfo
Input. Defines the size of the output parameter poNodeInfo, which must be large enough to hold status information from each database partition that is being rolled forward. In a single-partition database environment, this parameter should be set to 1. The value of this parameter should be the same as the number of database partition servers for which this API is being called.

piHistoryFile
History file.

iNumChngHistoryFile
Number of history files in list.

piChngHistoryFile
List of history files.

piComprLibrary
Input. Indicates the name of the external library to be used to perform decompression of the backup image if the image is compressed. The name must be a fully-qualified path referring to a file on the server. If the value is a null pointer or a pointer to an empty string, DB2 will attempt to use the library stored in the image. If the backup was not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore will fail.

piComprOptions
Input. Describes a block of binary data that will be passed to the initialization routine in the decompression library. DB2 will pass this string directly from the client to the server, so any issues of byte-reversal or code-page conversion will have to be handled by the compression library. If the first character of the data block is '@', the remainder of the data will be interpreted by DB2 as the name of a file residing on the server. DB2 will then replace the contents of piComprOptions and iComprOptionsSize with the contents and size of this file respectively and will pass these new values to the initialization routine instead.

iComprOptionsSize
Input. Represents the size of the block of data passed as piComprOptions. iComprOptionsSize shall be zero if and only if piComprOptions is a null pointer.

sqlu_histFile data structure parameters:

nodeNum
Input. Specifies which database partition this entry should be used for.

filenameLen
Input. Length in bytes of filename.

filename
Input. Path to the history file for this database partition. The path must end with a slash.

db2gRecoverStruct data structure specific parameters:

iSourceDBAliasLen
Specifies the length in bytes of the piSourceDBAlias parameter.

iUserNameLen
Specified the length in bytes of the piUsername parameter.

iPasswordLen
Specifies the length in bytes of the piPassword parameter.

iStopTimeLen
Specifies the length in bytes of the piStopTime parameter.

iOverflowLogPathLen
Specifies the length in bytes of the piOverflowLogPath parameter.

iHistoryFileLen
Specifies the length in bytes of the piHistoryFile parameter.

iComprLibraryLen
Input. Specifies the length in bytes of the name of the library specified in the piComprLibrary parameter. Set to zero if no library name is given.

Related tasks:

- "Using recover" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "SQLCA data structure" in *Administrative API Reference*
- "RECOVER DATABASE" on page 571
- "db2Rollforward - Roll forward a database" on page 767
- "db2Backup - Back up a database or table space" on page 665
- "db2Restore - Restore a database or table space" on page 756

db2Reorg - Reorganize an index or a table

Reorganizes a table or all indexes defined on a table by compacting the information and reconstructing the rows or index data to eliminate fragmented data.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

- dbadm
- CONTROL privilege on the table

Required connection:

Database

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Reorg (
            db2Uint32 versionNumber,
            void * pReorgStruct,
            struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2ReorgObject        reorgObject;
} db2ReorgStruct;

union db2ReorgObject
{
  struct db2ReorgTable             tableStruct;
  struct db2ReorgIndexesAll        indexesAllStruct;
};

typedef SQL_STRUCTURE db2ReorgTable
{
  char *pTableName;
  char *pOrderByIndex;
  char *pSysTempSpace;
  char *pLongTempSpace;
} db2ReorgTable;

typedef SQL_STRUCTURE db2ReorgIndexesAll
{
  char *pTableName;
  char *pIndexName;
} db2ReorgIndexesAll;

SQL_API_RC SQL_API_FN
  db2gReorg (
             db2Uint32 versionNumber,
             void * pReorgStruct,
             struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gReorgStruct
{
  db2Uint32 reorgType;
  db2Uint32 reorgFlags;
  db2int32 nodeListFlag;
  db2Uint32 numNodes;
  SQL_PDB_NODE_TYPE *pNodeList;
  union db2gReorgObject       reorgObject;
} db2gReorgStruct;

typedef SQL_STRUCTURE db2gReorgNodes
```

```
{
  SQL_PDB_NODE_TYPE nodeNum[SQL_PDB_MAX_NUM_NODE];
} db2gReorgNodes;

union db2gReorgObject
{
  struct db2gReorgTable           tableStruct;
  struct db2gReorgIndexesAll      indexesAllStruct;
};

typedef SQL_STRUCTURE db2gReorgTable
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 orderByIndexLen;
  char *pOrderByIndex;
  db2Uint32 sysTempSpaceLen;
  char *pSysTempSpace;
  db2Uint32 longTempSpaceLen;
  char *pLongTempSpace;
} db2gReorgTable;

typedef SQL_STRUCTURE db2gReorgIndexesAll
{
  db2Uint32 tableNameLen;
  char *pTableName;
  db2Uint32 indexNameLen;
  char *pIndexName;
} db2gReorgIndexesAll;
```

db2Reorg API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter, pReorgStruct.

pReorgStruct
Input. A pointer to the db2ReorgStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2ReorgStruct data structure parameters:

reorgType
Input. Specifies the type of reorganization. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2REORG_OBJ_TABLE_OFFLINE
Reorganize the table offline.

DB2REORG_OBJ_TABLE_INPLACE
Reorganize the table inplace.

DB2REORG_OBJ_INDEXESALL
Reorganize all indexes.

DB2REORG_OBJ_INDEX
Reorganize one index.

reorgFlags
Input. Reorganization options. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2REORG_OPTION_NONE
Default action.

DB2REORG_LONGLOB
Reorganize long fields and lobs, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the reorgType .

DB2REORG_INDEXSCAN
Recluster utilizing index scan, used when DB2REORG_OBJ_TABLE_OFFLINE is specified as the reorgType .

DB2REORG_START_ONLINE
Start online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the reorgType .

DB2REORG_PAUSE_ONLINE
Pause an existing online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the reorgType .

DB2REORG_STOP_ONLINE
Stop an existing online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the reorgType .

DB2REORG_RESUME_ONLINE
Resume a paused online reorganization, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the reorgType .

DB2REORG_NOTRUNCATE_ONLINE
Do not perform table truncation, used when DB2REORG_OBJ_TABLE_INPLACE is specified as the reorgType .

DB2REORG_ALLOW_NONE
No read or write access to the table. This parameter is not supported when DB2REORG_OBJ_TABLE_INPLACE is specified as the reorgType .

DB2REORG_ALLOW_WRITE
Allow read and write access to the table. This parameter is not supported when DB2REORG_OBJ_TABLE_OFFLINE is specified as the reorgType .

DB2REORG_ALLOW_READ
Allow only read access to the table.

DB2REORG_CLEANUP_NONE
No clean up is required, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the reorgType.

DB2REORG_CLEANUP_ALL
Clean up the committed pseudo deleted keys and committed pseudo empty pages, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the reorgType .

DB2REORG_CLEANUP_PAGES
Clean up committed pseudo empty pages only, but do not clean up pseudo deleted keys on pages that are not pseudo empty, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the reorgType.

DB2REORG_CONVERT_NONE
No conversion is required, used when DB2REORG_OBJ_INDEXESALL or DB2REORG_OBJ_INDEX are specified as the reorgType.

DB2REORG_CONVERT
Convert to type 2 index, used when DB2REORG_OBJ_INDEXESALL is specified as the reorgType.

DB2REORG_RESETDICTIONARY
If the COMPRESS attribute for the table is YES then a new row compression dictionary is built. All the rows processed during reorganization are subject to compression using this new dictionary. This dictionary replaces any previous dictionary. If the COMPRESS attribute for the table is NO and the table does have an existing compression dictionary then reorg processing will remove the dictionary and all rows in the newly reorganized table will be in non-compressed format. This parameter is only supported for the DB2REORG_OBJ_TABLE_OFFLINE reorgType.

DB2REORG_KEEPDICTIONARY
If the COMPRESS attribute for the table is YES and a dictionary exists, it is kept. If the COMPRESS attribute for the table is YES and a dictionary does not exist, one is built, as the option defaults to DB2REORG_RESETDICTIONARY in that case. All rows processed by reorganization are subject to compression. If the COMPRESS attribute for the table is NO, the dictionary will be retained (if one existed), and all rows in the newly reorganized table will be in non-compressed format. This parameter is only supported for the DB2REORG_OBJ_TABLE_OFFLINE reorgType.

nodeListFlag
Input. Specifies which nodes to reorganize. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2REORG_NODE_LIST
Submit to all nodes in the nodelist array.

DB2REORG_ALL_NODES
Submit to all nodes in the database partition group.

DB2REORG_ALL_EXCEPT
Submit to all nodes except the ones specified by the nodelist parameter.

numNodes
Input. Number of nodes in the nodelist array.

pNodeList
A pointer to the array of node numbers.

reorgObject
Input. Specifies the type of object to be reorganized.

db2ReorgObject union parameters:

tableStruct
Specifies the options for a table reorganization.

indexesAllStruct
Specifies the options for an index reorganization.

db2ReorgTable data structure parameters:

pTableName
Input. Specifies the name of the table to reorganize.

pOrderByIndex
Input. Specifies the index to order the table by.

pSysTempSpace
Input. Specifies the system temporary table space where temporary objects are created. The REORG command may expand rows in cases where a column is added to a table (i.e. from ALTER TABLE ADD COLUMN) and the rows were inserted before the column was added. For a non-partitioned table, this parameter must specify a table space with enough room to create the new table object. A partitioned table is reorganized a single data partition at a time. In this case, there must be enough free space in the table space to hold the largest data partition of the table.

If this parameter is not specified for a non-partitioned table the table space the table resides in is used. If this parameter is not specified for a partitioned table, the table space where each data partition is located is used for temporary storage of that data partition. There must be enough free space in each data partition's table space to hold a copy of the data partition.

pLongTempSpace
Input. Specifies the temporary table space to create long objects (LONG VARCHAR and LOB columns) in during table reorginization. If the pSysTempSpace parameter is not specified, this parameter is ignored. If this parameter is not specified, but the pSysTempSpace parameter is specified, then DB2 will create the long data objects in the table space specified by the pSysTempSpace parameter, unless the page sizes differ.

When page sizes differ, if pSysTempSpace is specified, but this parameter is not, DB2 will attempt to find an existing table space with a matching page size to create the long objects in.

db2ReorgIndexesAll data structure parameters:

pTableName
Input. Specifies the name of the table for index reorganization. If DB2REORG_OBJ_INDEX is specified as the reorgType, the pTableName parameter is not required and can be NULL. However, if the pTableName parameter is specified, it must be the table on which the index is defined.

pIndexName
Input. Specifies the name of the index to reorganize. This parameter is used only when the reorgType parameter is set to a value of DB2REORG_OBJ_INDEX otherwise set pIndexName parameter to NULL.

db2gReorgTable data structure specific parameters:

tableNameLen
Input. Specifies the length in bytes of pTableName.

orderByIndexLen
Input. Specifies the length in byte of pOrderByIndex.

sysTempSpaceLen
Input. Specifies the length in bytes of pSysTempSpace.

longTempSpaceLen
Input. Specifies the length of the name stored in the pLongTempSpace parameter.

db2gReorgIndexesAll data structure specific parameters:

tableNameLen
Input. Specifies the length in bytes of pTableName.

indexNameLen
Input. Specifies the length in bytes of the pIndexName parameter.

Usage notes:

- Performance of table access, index scans, and the effectiveness of index page prefetching can be adversely affected when the table data has been modified many times, becoming fragmented and unclustered. Use REORGCHK to determine whether a table or its indexes are candidates for reorganizing. All work will be committed and all open cursors will be closed during reorg processing. After reorganizing a table or its indexes, use db2Runstats to update the statistics and sqlarbnd to rebind the packages that use this table.
- If the table data is distributed onto several nodes and the reorganization fails on any of the affected nodes, then only the failing nodes will have the reorganization rolled back. If table reorganization is not successful, temporary files should not be deleted. The database manager uses these files to recover the database.
- For table reorganization, if the name of an index is specified, the database manager reorganizes the data according to the order in the index. To maximize performance, specify an index that is often used in SQL queries. If the name of an index is not specified, and if a clustering index exists, the data will be ordered according to the clustering index.
- The PCTFREE value of a table determines the amount of free space designated per page. If the value has not been set, the utility will fill up as much space as possible on each page.
- To complete a table space roll-forward recovery following a table reorganization, both data and LONG table spaces must be roll-forward enabled.
- If the table contains LOB columns not defined with the COMPACT option, the LOB DATA storage object can be significantly larger following table reorganization. This can be a result of the order in which the rows were reorganized, and the types of table spaces used (SMS/DMS).
- The following table illustrates the default table access chosen based on the type of reorg and table:

Table 133. Default table access chosen based on the type of reorg and table

Type of reorg and applicable flags which can affect the default table access		Access mode chosen for each table type	
reorgType	**reorgFlags (if applicable)**	**Non-partitioned Table**	**Partitioned Table**
DB2REORG_OBJ_TABLE_ OFFLINE		DB2REORG_ALLOW_READ	DB2REORG_ALLOW_NONE
DB2REORG_OBJ_TABLE_ INPLACE		DB2REORG_ALLOW_WRITE	N/A
DB2REORG_OBJ_INDEXESALL		DB2REORG_ALLOW_READ	DB2REORG_ALLOW_NONE
DB2REORG_OBJ_INDEXESALL	DB2REORG_CLEANUP_ALL, DB2REORG_CLEANUP_ PAGES	DB2REORG_ALLOW_READ	DB2REORG_ALLOW_READ
DB2REORG_OBJ_INDEX		N/A	DB2REORG_ALLOW_READ
DB2REORG_OBJ_INDEX	DB2REORG_CLEANUP_ALL, DB2REORG_CLEANUP_ PAGES	N/A	DB2REORG_ALLOW_READ

N/A: Not applicable at this time since it is not supported.

Note: Some access modes may not be supported on certain types of tables or indexes. In these cases and where possible, the least restrictive access mode is used. (The most restrictive access mode being DB2REORG_ALLOW_NONE, followed by DB2REORG_ALLOW_READ, and then DB2REORG_ALLOW_WRITE, which is the least restrictive). As support for existing table or index types change, or new table or index types are provided, the default can change from a more restrictive access mode to a less restrictive mode. The least restrictive mode chosen for the default will not go beyond DB2REORG_ALLOW_READ when the reorgType is not DB2REORG_OBJ_TABLE_INPLACE. The default access mode is chosen when none of the DB2REORG_ALLOW_NONE, DB2REORG_ALLOW_READ, or DB2REORG_ALLOW_WRITE flags are specified.

- When reorganizing indexes, use the access option to allow other transactions either read only or read-write access to the table. There is a brief lock-out period when the reorganized index(es) are being made available during which no access to the table is allowed.
- If an index reorganization with allow read or allow write access fails because the indexes need to be rebuilt, the reorganization will be switched to allow no access and carry on. A message will be written to both the administration notification log and the diagnostics log to warn you about the change in the access mode. For an index reorganization of a partitioned table, any indexes that need to be rebuilt are rebuilt offline and then the index you specified is reorganized, assuming it was not already rebuilt. This reorganization uses the access mode specified by you. A message will be written to the administration notification log and the diagnostics log to warn you that the indexes are being rebuilt offline.
- For non-inplace table reorganization, if neither DB2REORG_RESETDICTIONARY or DB2REORG_KEEPDICTIONARY is specified, the default is DB2REORG_KEEPDICTIONARY.
- If an index reorganization with no access fails, some or all indexes will not be available and will be rebuilt on the next table access.
- This API cannot be used with:
  - views or an index that is based on an index extension
  - a DMS table while an online backup of a table space in which the table resides is being performed
  - declared temporary tables

Related concepts:

- "Table reorganization" in *Performance Guide*

Related reference:

- "REORG INDEXES/TABLE " on page 580
- "REORGCHK command" in *Command Reference*
- "ADMIN_CMD procedure ÔÇô Run administrative commands" in *Administrative SQL Routines and Views*
- "db2Runstats API - Update statistics for tables and indexes" in *Administrative API Reference*
- "sqlarbnd - Rebind package" on page 833
- "SQLCA data structure" in *Administrative API Reference*

Related samples:

- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"

- "tbreorg.sqc -- How to reorganize a table and update its statistics (C)"
- "tbreorg.sqC -- How to reorganize a table and update its statistics (C++)"

db2Restore - Restore a database or table space

Recreates a damaged or corrupted database that has been backed up using the db2Backup API. The restored database is in the same state it was in when the backup copy was made. This utility can also restore to a database with a name different from the database name in the backup image (in addition to being able to restore to a new database).

This utility can also be used to restore DB2 databases created in the two previous releases.

This utility can also restore from a table space level backup, or restore table spaces from within a database backup image.

Scope:

This API only affects the database partition from which it is called.

Authorization:

To restore to an existing database, one of the following:
- *sysadm*
- *sysctrl*
- *sysmaint*

To restore to a new database, one of the following:
- *sysadm*
- *sysctrl*

Required connection:

Database, to restore to an existing database. This API automatically establishes a connection to the specified database and will release the connection when the restore operation finishes.

Instance and *database*, to restore to a new database. The instance attachment is required to create the database.

To restore to a new database at an instance different from the current instance (as defined by the value of the `DB2INSTANCE` environment variable), it is necessary to first attach to the instance where the new database will reside.

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Restore (
       db2Uint32 versionNumber,
       void * pDB2RestoreStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RestoreStruct
{
  char *piSourceDBAlias;
  char *piTargetDBAlias;
  char oApplicationId[SQLU_APPLID_LEN+1];
  char *piTimestamp;
  char *piTargetDBPath;
  char *piReportFile;
  struct db2TablespaceStruct *piTablespaceList;
  struct db2MediaListStruct *piMediaList;
  char *piUsername;
  char *piPassword;
  char *piNewLogPath;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 iParallelism;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iCallerAction;
  db2Uint32 iOptions;
  char *piComprLibrary;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  char *piLogTarget;
  struct db2StoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
} db2RestoreStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                        **tablespaces;
  db2Uint32 numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                        **locations;
  db2Uint32 numLocations;
  char locationType;
} db2MediaListStruct;

typedef SQL_STRUCTURE db2StoragePathsStruct
{
  char                        **storagePaths;
  db2Uint32 numStoragePaths;
} db2StoragePathsStruct;

SQL_API_RC SQL_API_FN
  db2gRestore (
       db2Uint32 versionNumber,
       void * pDB2gRestoreStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRestoreStruct
{
  char *piSourceDBAlias;
  db2Uint32 iSourceDBAliasLen;
  char *piTargetDBAlias;
  db2Uint32 iTargetDBAliasLen;
  char *poApplicationId;
  db2Uint32 iApplicationIdLen;
  char *piTimestamp;
  db2Uint32 iTimestampLen;
  char *piTargetDBPath;
  db2Uint32 iTargetDBPathLen;
  char *piReportFile;
  db2Uint32 iReportFileLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct *piMediaList;
  char *piUsername;
  db2Uint32 iUsernameLen;
  char *piPassword;
  db2Uint32 iPasswordLen;
  char *piNewLogPath;
  db2Uint32 iNewLogPathLen;
  void *piVendorOptions;
  db2Uint32 iVendorOptionsSize;
  db2Uint32 iParallelism;
  db2Uint32 iBufferSize;
  db2Uint32 iNumBuffers;
  db2Uint32 iCallerAction;
  db2Uint32 iOptions;
  char *piComprLibrary;
  db2Uint32 iComprLibraryLen;
  void *piComprOptions;
  db2Uint32 iComprOptionsSize;
  char *piLogTarget;
  db2Uint32 iLogTargetLen;
  struct db2gStoragePathsStruct *piStoragePaths;
  char *piRedirectScript;
  db2Uint32 iRedirectScriptLen;
} db2gRestoreStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char *tablespaces;
  db2Uint32 numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char *locations;
  db2Uint32 numLocations;
  char locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2gStoragePathsStruct
{
  struct db2Char *storagePaths;
  db2Uint32 numStoragePaths;
} db2gStoragePathsStruct;

typedef SQL_STRUCTURE db2Char
{
   char *pioData;
   db2Uint32 iLength;
   db2Uint32 oLength;
} db2Char;
```

db2Restore API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pDB2RestoreStruct.

pDB2RestoreStruct
Input. A pointer to the db2RestoreStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RestoreStruct data structure parameters:

piSourceDBAlias
Input. A string containing the database alias of the source database backup image.

piTargetDBAlias
Input. A string containing the target database alias. If this parameter is null, the value of the `piSourceDBAlias` parameter will be used.

oApplicationId
Output. The API will return a string identifying the agent servicing the application. Can be used to obtain information about the progress of the backup operation using the database monitor.

piTimestamp
Input. A string representing the time stamp of the backup image. This field is optional if there is only one backup image in the source specified.

piTargetDBPath
Input. A string containing the relative or fully qualified name of the target database directory on the server. Used if a new database is to be created for the restored backup; otherwise not used.

piReportFile
Input. The file name, if specified, must be fully qualified. The datalinks files that become unlinked during a restore (as a result of a fast reconcile) will be reported.

piTablespaceList
Input. List of table spaces to be restored. Used when restoring a subset of table spaces from a database or table space backup image. For rebuild cases, this can be an include list or exclude list of table spaces used to rebuild your database. See the `DB2TablespaceStruct` structure. The following restrictions apply:

- The database must be recoverable (for non-rebuild cases only); that is, log retain or user exits must be enabled.
- The database being restored to must be the same database that was used to create the backup image. That is, table spaces can not be added to a database through the table space restore function.
- The rollforward utility will ensure that table spaces restored in a partitioned database environment are synchronized with any other database partition containing the same table spaces. If a table space restore operation is requested and the `piTablespaceList` is `NULL`, the restore utility will attempt to restore all of the table spaces in the backup image.
- When restoring a table space that has been renamed since it was backed up, the new table space name must be used in the restore command. If the old table space name is used, it will not be found.
- In the case of rebuild, the list must be given for 3 of the 5 rebuild types: `DB2RESTORE_ALL_TBSP_IN_DB_EXC`, `DB2RESTORE_ALL_TBSP_IN_IMG_EXC` and `DB2RESTORE_ALL_TBSP_IN_LIST`

piMediaList
Input. Source media for the backup image. The information provided depends on the value of the locationType parameter. The valid values for locationType parameter (defined in sqlutil header file, located in the include directory) are:

- SQLU_LOCAL_MEDIA - Local devices (a combination of tapes, disks, or diskettes).
- SQLU_XBSA_MEDIA - XBSA interface. Backup Services APIs (XBSA) are an open application programming interface for applications or facilities needing data storage management for backup or archiving purposes.
- SQLU_TSM_MEDIA - TSM. If the locations pointer is set to NULL, the TSM shared library provided with DB2 is used. If a different version of the TSM shared library is desired, use SQLU_OTHER_MEDIA and provide the shared library name.
- SQLU_OTHER_MEDIA - Vendor product. Provide the shared library name in the locations field.
- SQLU_USER_EXIT - User exit. No additional input is required (only available when server is on OS/2).

piUsername
Input. A string containing the user name to be used when attempting a connection. Can be NULL.

piPassword
Input. A string containing the password to be used with the user name. Can be NULL.

piNewLogPath
Input. A string representing the path to be used for logging after the restore has completed. If this field is null the default log path will be used.

piVendorOptions
Input. Used to pass information from the application to the vendor functions. This data structure must be flat; that is, no level of indirection is supported. Note that byte-reversal is not done, and the code page is not checked for this data.

iVendorOptionsSize
Input. The length in bytes of the piVendorOptions parameter, which cannot exceed 65535 bytes.

iParallelism
Input. Degree of parallelism (number of buffer manipulators). Minimum is 1. Maximum is 1024.

iBufferSize
Input. Backup buffer size in 4 KB allocation units (pages). Minimum is 8 units. The size entered for a restore must be equal to or an integer multiple of the buffer size used to produce the backup image.

iNumBuffers
Input. Specifies number of restore buffers to be used.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

- DB2RESTORE_RESTORE - Start the restore operation.
- DB2RESTORE_NOINTERRUPT - Start the restore. Specifies that the restore will run unattended, and that scenarios which normally require user intervention will either be attempted without first returning to the caller, or will generate an error. Use this caller action, for example, if it is known that all of the media required for the restore have been mounted, and utility prompts are not desired.

- `DB2RESTORE_CONTINUE` - Continue the restore after the user has performed some action requested by the utility (mount a new tape, for example).
- `DB2RESTORE_TERMINATE` - Terminate the restore after the user has failed to perform some action requested by the utility.
- `DB2RESTORE_DEVICE_TERMINATE` - Remove a particular device from the list of devices used by restore. When a particular device has exhausted its input, restore will return a warning to the caller. Call restore again with this caller action to remove the device which generated the warning from the list of devices being used.
- `DB2RESTORE_PARM_CHK` - Used to validate parameters without performing a restore. This option does not terminate the database connection after the call returns. After a successful return of this call, it is expected that the user will issue another call to this API with the iCallerAction parameter set to the value DB2RESTORE_CONTINUE to continue with the restore.
- `DB2RESTORE_PARM_CHK_ONLY` - Used to validate parameters without performing a restore. Before this call returns, the database connection established by this call is terminated, and no subsequent call is required.
- `DB2RESTORE_TERMINATE_INCRE` - Terminate an incremental restore operation before completion.
- `DB2RESTORE_RESTORE_STORDEF` - Initial call. Table space container redefinition requested.
- `DB2RESTORE_STORDEF_NOINTERRUPT` - Initial call. The restore will run uninterrupted. Table space container redefinition requested.

iOptions

Input. A bitmap of restore properties. The options are to be combined using the bitwise OR operator to produce a value for iOptions. Valid values (defined in db2ApiDf header file, located in the include directory) are:

- `DB2RESTORE_OFFLINE` - Perform an offline restore operation.
- `DB2RESTORE_ONLINE` - Perform an online restore operation.
- `DB2RESTORE_DB` - Restore all table spaces in the database. This must be run offline.
- `DB2RESTORE_TABLESPACE` - Restore only the table spaces listed in the piTablespaceList parameter from the backup image. This can be online or offline.
- `DB2RESTORE_HISTORY` - Restore only the history file.
- `DB2RESTORE_COMPR_LIB` - Indicates that the compression library is to be restored. This option cannot be used simultaneously with any other type of restore process. If the object exists in the backup image, it will be restored into the database directory. If the object does not exist in the backup image, the restore operation will fail.
- `DB2RESTORE_LOGS` - Specifies that only the set of log files contained in the backup image are to be restored. If the backup image does not include log files, the restore operation will fail. If this option is specified, the piLogTarget parameter must also be specified.
- `DB2RESTORE_INCREMENTAL` - Perform a manual cumulative restore operation.

- DB2RESTORE_AUTOMATIC - Perform an automatic cumulative (incremental) restore operation. Must be specified with DB2RESTORE_INCREMENTAL.
- DB2RESTORE_DATALINK - Perform reconciliation operations. Tables with a defined DATALINK column must have RECOVERY YES option specified.
- DB2RESTORE_NODATALINK - Do not perform reconciliation operations. Tables with DATALINK data type columns are placed into DataLink_Roconcile_pending (DRP) state. Tables with a defined DATALINK column must have the RECOVERY YES option specified.
- DB2RESTORE_ROLLFWD - Place the database in rollforward pending state after it has been successfully restored.
- DB2RESTORE_NOROLLFWD - Do not place the database in rollforward pending state after it has been successfully restored. This cannot be specified for backups taken online or for table space level restores. If, following a successful restore, the database is in roll-forward pending state, the db2Rollforward API must be called before the database can be used.
- DB2RESTORE_GENERATE_SCRIPT - Create a script, that can be used to perform a redirected restore. piRedirectScript must contain a valid file name. The iCallerAction need to be either DB2RESTORE_RESTORE_STORDEF or DB2RESTORE_STORDEF_NOINTERRUPT.

The following values should be used for rebuild operations only:

- DB2RESTORE_ALL_TBSP_IN_DB - Restores the database with all the table spaces known to the database at the time of the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_DB_EXC - Restores the database with all the table spaces known to the database at the time of the image being restored except for those specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG - Restores the database with only the table spaces in the image being restored. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_IMG_EXC - Restores the database with only the table spaces in the image being restored except for those specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.
- DB2RESTORE_ALL_TBSP_IN_LIST - Restores the database with only the table spaces specified in the list pointed to by the piTablespaceList parameter. This rebuild overwrites a database if it already exists.

NOTE: If the backup image is of a recoverable database then WITHOUT ROLLING FORWARD (DB2RESTORE_NOROLLFWD) cannot be specified with any of the above rebuild actions.

piComprLibrary
Input. Indicates the name of the external library to use to decompress the backup image if the image is compressed. The name must be a fully-qualified path that refers to a file on the server. If the value is a null pointer or a pointer to an empty string, the DB2 database system attempts

to use the library stored in the image. If the backup is not compressed, the value of this parameter will be ignored. If the specified library is not found, the restore operation will fail.

piComprOptions
Input. This API parameter describes a block of binary data that will be passed to the initialization routine in the decompression library. The DB2 database system passes this string directly from the client to the server, so any issues of byte-reversal or code-page conversion must be handled by the compression library. If the first character of the data block is '@', the remainder of the data is interpreted as the name of a file residing on the server. The DB2 database system then replaces the contents of the piComprOptions and iComprOptionsSize parameters with the contents and size of this file and passes these new values to the initialization routine.

iComprOptionsSize
Input. A four-byte unsigned integer that represents the size of the block of data passed as piComprOptions. The iComprOptionsSize parameter should be zero if and only if the piComprOptions value is a null pointer.

piLogTarget
Input. Specifies the absolute path of a directory on the database server that must be used as the target directory for extracting log files from a backup image. If this parameter is specified, any log files included in the backup image are extracted into the target directory. If this parameter is not specified, log files included in the backup image are not extracted. To extract only the log files from the backup image, DB2RESTORE_LOGS value should be passed to the iOptions parameter.

piStoragePaths
Input. A structure containing fields that describe a list of storage paths used for automatic storage. Set this to NULL if automatic storage is not enabled for the database.

piRedirectScript
Input. The file name for the redirect restore script that will be created on client side. The file name can be specified relative or absolute. The iOptions field need to have the DB2RESTORE_GENERATE_SCRIPT bit set.

db2TablespaceStruct data structure specific parameters:

tablespaces
Input. A pointer to the list of table spaces to be backed up. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numTablespaces
Input. Number of entries in the tablespaces parameter.

db2MediaListStruct data structure parameters:

locations
Input. A pointer to the list of media locations. For C, the list is null-terminated strings. In the generic case, it is a list of db2Char structures.

numLocations
Input. The number of entries in the locations parameter.

locationType
Input. A character indicating the media type. Valid values (defined in sqlutil header file, located in the include directory.) are:
- `SQLU_LOCAL_MEDIA` - Local devices (tapes, disks, diskettes, or named pipes).
- `SQLI_XBSA_MEDIA` - XBSA interface.
- `SQLU_TSM_MEDIA` - Tivoli Storage Manager.
- `SQLU_OTHER_MEDIA` - Vendor library.
- `SQLU_USER_EXIT` - User exit (only available when the server is on OS/2).

db2StoragePathsStruct data structure parameters:

storagePaths
Input. An array of strings containing fully qualified names of storage paths on the server that will be used for automatic storage table spaces. In a multi-partition database the same storage paths are used on all database partitions. If a multi-partiton database is being restored with new storage paths, then the catalog partition must be restored before any other database partitions are restored.

numStoragePaths
Input. The number of storage paths in the storagePaths parameter of the db2StoragePathsStruct structure.

db2gRestoreStruct data structure specific parameters:

iSourceDBAliasLen
Input. Specifies the length in bytes of the piSourceDBAlias parameter.

iTargetDBAliasLen
Input. Specifies the length in bytes of the piTargetDBAlias parameter.

iApplicationIdLen
Input. Specifies the length in bytes of the poApplicatoinId parameter. Should be equal to SQLU_APPLID_LEN + 1. The constant SQLU_APPLID_LEN is defined in sqlutil header file that is located in the include directory.

iTimestampLen
Input. Specifies the length in bytes of the piTimestamp parameter.

iTargetDBPathLen
Input. Specifies the length in bytes of the piTargetDBPath parameter.

iReportFileLen
Input. Specifies the length in bytes of the piReportFile parameter.

iUsernameLen
Input. Specifies the length in bytes of the piUsername parameter. Set to zero if no user name is provided.

iPasswordLen
Input. Specifies the length in bytes of the piPassword parameter. Set to zero if no password is provided.

iNewLogPathLen
Input. Specifies the length in bytes of the piNewLogPath parameter.

iLogTargetLen
Input. Specifies the length in bytes of the piLogTarget parameter.

iRedirectScriptLen
Input. A four-byte unsigned integer representing the length in bytes of the name of the library specified in piRedirectScript. Set to zero if no script name is given.

db2Char data structure parameters:

pioData
A pointer to a character data buffer. If NULL, no data will be returned.

iLength
Input. The size of the pioData buffer.

oLength
Output. The number of valid characters of data in the pioData buffer.

Usage notes:

- For offline restore, this utility connects to the database in exclusive mode. The utility fails if any application, including the calling application, is already connected to the database that is being restored. In addition, the request will fail if the restore utility is being used to perform the restore, and any application, including the calling application, is already connected to any database on the same workstation. If the connect is successful, the API locks out other applications until the restore is completed.
- The current database configuration file will not be replaced by the backup copy unless it is unusable. In this case, if the file is replaced, a warning message is returned.
- The database or table space must have been backed up using the db2Backup API.
- If the caller action value is DB2RESTORE_NOINTERRUPT, the restore continues without prompting the application. If the caller action value is DB2RESTORE_RESTORE, and the utility is restoring to an existing database, the utility returns control to the application with a message requesting some user interaction. After handling the user interaction, the application calls RESTORE DATABASE again, with the caller action value set to indicate whether processing is to continue (DB2RESTORE_CONTINUE) or terminate (DB2RESTORE_TERMINATE) on the subsequent call. The utility finishes processing, and returns an SQLCODE in the sqlca.
- To close a device when finished, set the caller action value to DB2RESTORE_DEVICE_TERMINATE. If, for example, a user is restoring from 3 tape volumes using 2 tape devices, and one of the tapes has been restored, the application obtains control from the API with an SQLCODE indicating end of tape. The application can prompt the user to mount another tape, and if the user indicates ″no more″, return to the API with caller action value SQLUD_DEVICE_TERMINATE to signal end of the media device. The device driver will be terminated, but the rest of the devices involved in the restore will continue to have their input processed until all segments of the restore set have been restored (the number of segments in the restore set is placed on the last media device during the backup process). This caller action can be used with devices other than tape (vendor supported devices).
- To perform a parameter check before returning to the application, set caller action value to DB2RESTORE_PARM_CHK.
- Set caller action value to DB2RESTORE_RESTORE_STORDEF when performing a redirected restore; used in conjunction with the sqlbstsc API.

- If a system failure occurs during a critical stage of restoring a database, the user will not be able to successfully connect to the database until a successful restore is performed. This condition will be detected when the connection is attempted, and an error message is returned. If the backed-up database is not configured for roll-forward recovery, and there is a usable current configuration file with either of these parameters enabled, following the restore, the user will be required to either take a new backup of the database, or disable the log retain and user exit parameters before connecting to the database.
- Although the restored database will not be dropped (unless restoring to a nonexistent database), if the restore fails, it will not be usable.
- If the restore type specifies that the history file in the backup is to be restored, it will be restored over the existing history file for the database, effectively erasing any changes made to the history file after the backup that is being restored. If this is undesirable, restore the history file to a new or test database so that its contents can be viewed without destroying any updates that have taken place.
- If, at the time of the backup operation, the database was enabled for roll forward recovery, the database can be brought to the state it was in prior to the occurrence of the damage or corruption by issuing db2Rollforward after successful execution of db2Restore. If the database is recoverable, it will default to roll forward pending state after the completion of the restore.
- If the database backup image is taken offline, and the caller does not want to roll forward the database after the restore, the DB2RESTORE_NOROLLFWD option can be used for the restore. This results in the database being useable immediately after the restore. If the backup image is taken online, the caller must roll forward through the corresponding log records at the completion of the restore.
- To restore log files from a backup image that contains them, the LOGTARGET option must be specified, assuming a fully qualified and valid path exists on the DB2 server. If those conditions are satisfied, the restore utility writes the log files from the image to the target path. If LOGTARGET is specified during a restoration of a backup image that does not include logs, the restore operation returns an error before attempting to restore any table space data. A restore operation also fails with an error if an invalid or read-only LOGTARGET path is specified.
- If any log files exist in the LOGTARGET path at the time the RESTORE command is issued, a warning prompt is returned to user. This warning is not returned if WITHOUT PROMPTING is specified.
- During a restore operation in which a LOGTARGET is specified, if any log file cannot be extracted, the restore operation fails and returns an error. If any of the log files being extracted from the backup image have the same name as an existing file in the LOGTARGET path, the restore operation fails and an error is returned. The restore utility does not overwrite existing log files in the LOGTARGET directory.
- You can restore only the saved log set from a backup image. To indicate that only the log files are to be restored, specify the LOGS option in addition to the LOGTARGET path. Specifying the LOGS option without a LOGTARGET path results in an error. If any problem occurs while restoring log files in this mode the restore operation terminates immediately and an error is returned.
- During an automatic incremental restore operation, only the logs included in the target image of the restore operation are retrieved from the backup image. Any logs that are included in intermediate images that are referenced during the incremental restore process are not extracted from those intermediate backup images. During a manual incremental restore operation, the LOGTARGET path should be specified only with the final restore command.

- If a backup is compressed, the DB2 database system detects this state and automatically decompresses the data before restoring it. If a library is specified on the db2Restore API, it is used for decompressing the data. If a library is not specified on the db2Restore API, the library stored in the backup image is used. And if there is no library stored in the backup image, the data cannot be decompressed and the restore operation fails.
- If the compression library is being restored from a backup image (either explicitly by specifying the DB2RESTORE_COMPR_LIB restore type or implicitly by performing a normal restoration of a compressed backup), the restore operation must be done on the same platform and operating system that the backup was taken on. If the platforms are different, the restore operation will fail, even when the DB2 database system normally supports cross-platform restore operations involving the two systems.
- If restoring a database that is enabled for automatic storage, the storage paths associated with the database can be redefined or they can remain as they were previously. To keep the storage path definitions as is, do not provide any storage paths as part of the restore operation. Otherwise, specify a new set of storage paths to associate with the database. Automatic storage table spaces will be automatically redirected to the new storage paths during the restore operation.

Related tasks:
- "Using restore" in *Data Recovery and High Availability Guide and Reference*

Related reference:
- "sqlemgdb - Migrate previous version of DB2 database to current version" on page 815
- "SQLCA data structure" in *Administrative API Reference*
- "RESTORE DATABASE " on page 591
- "db2Rollforward - Roll forward a database" on page 767
- "db2Backup - Back up a database or table space" on page 665
- "db2Recover - Restore and roll forward a database" on page 743

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2Rollforward - Roll forward a database

Recovers a database by applying transactions recorded in the database log files. Called after a database or a table space backup has been restored, or if any table spaces have been taken offline by the database due to a media error. The database must be recoverable (that is, either the logarchmeth1 database configuration parameter or the logarchmeth2 database configuration parameter must be set to a value other than OFF) before the database can be recovered with rollforward recovery.

Scope:

In a partitioned database environment, this API can only be called from the catalog partition. A database or table space rollforward call specifying a point-in-time affects all database partition servers that are listed in the db2nodes.cfg file. A database or table space rollforward call specifying end of logs affects the database partition servers that are specified. If no database partition servers are specified, it affects all database partition servers that are listed in the db2nodes.cfg file; if no roll forward is needed on a particular database partition server, that database partition server is ignored.

For partitioned tables, you are also required to roll forward related table spaces to the same point in time. Related table spaces contain attached, detached, and dropped data partitions or indexes of a table. Rollforward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint

Required connection:

None. This API establishes a database connection.

API include file:

`db2ApiDf.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2Rollforward (
       db2Uint32 versionNumber,
       void * pDB2RollforwardStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
  struct db2RfwdInputStruct *piRfwdInput;
  struct db2RfwdOutputStruct *poRfwdOutput;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
  sqluint32 iVersion;
  char *piDbAlias;
  db2Uint32 iCallerAction;
  char *piStopTime;
  char *piUserName;
  char *piPassword;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2Uint32 iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piDroppedTblID;
  char *piExportDir;
  db2Uint32 iRollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char *poApplicationId;
  sqlint32 *poNumReplies;
  struct sqlurf_info *poNodeInfo;
  db2Uint32 oRollforwardFlags;
} db2RfwdOutputStruct;

SQL_STRUCTURE sqlurf_newlogpath
{
   SQL_PDB_NODE_TYPE nodenum;
   unsigned short  pathlen;
   char logpath[SQL_LOGPATH_SZ+SQL_LOGFILE_NAME_SZ+1];
};

typedef SQL_STRUCTURE sqlu_tablespace_bkrst_list
{
   sqlint32 num_entry;
   struct sqlu_tablespace_entry *tablespace;
} sqlu_tablespace_bkrst_list;

typedef SQL_STRUCTURE sqlu_tablespace_entry
{
   sqluint32 reserve_len;
   char tablespace_entry[SQLU_MAX_TBS_NAME_LEN+1];
   char filler[1];
} sqlu_tablespace_entry;

SQL_STRUCTURE sqlurf_info
{
   SQL_PDB_NODE_TYPE nodenum;
   sqlint32 state;
   unsigned char   nextarclog[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   firstarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastarcdel[SQLUM_ARCHIVE_FILE_LEN+1];
   unsigned char   lastcommit[SQLUM_TIMESTAMP_LEN+1];
};

SQL_API_RC SQL_API_FN
  db2gRollforward (
       db2Uint32 versionNumber,
       void * pDB2gRollforwardStruct,
       struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct *piRfwdInput;
  struct db2RfwdOutputStruct *poRfwdOutput;
} db2gRollforwardStruct;

typedef SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32 iDbAliasLen;
  db2Uint32 iStopTimeLen;
  db2Uint32 iUserNameLen;
  db2Uint32 iPasswordLen;
  db2Uint32 iOvrflwLogPathLen;
  db2Uint32 iDroppedTblIDLen;
  db2Uint32 iExportDirLen;
  sqluint32 iVersion;
  char *piDbAlias;
  db2Uint32 iCallerAction;
  char *piStopTime;
  char *piUserName;
  char *piPassword;
  char *piOverflowLogPath;
  db2Uint32 iNumChngLgOvrflw;
  struct sqlurf_newlogpath *piChngLogOvrflw;
  db2Uint32 iConnectMode;
  struct sqlu_tablespace_bkrst_list *piTablespaceList;
  db2int32 iAllNodeFlag;
  db2int32 iNumNodes;
  SQL_PDB_NODE_TYPE *piNodeList;
  db2int32 iNumNodeInfo;
  char *piDroppedTblID;
  char *piExportDir;
  db2Uint32 iRollforwardFlags;
} db2gRfwdInputStruct;
```

db2Rollforward API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter.

pDB2RollforwardStruct
Input. A pointer to the db2RollforwardStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2RollforwardStruct data structure parameters:

piRfwdInput
Input. A pointer to the db2RfwdInputStruct structure.

poRfwdOutput
Output. A pointer to the db2RfwdOutputStruct structure.

db2RfwdInputStruct data structure parameters:

iVersion
Input. The version ID of the rollforward parameters. It is defined as SQLUM_RFWD_VERSION.

piDbAlias
Input. A string containing the database alias. This is the alias that is cataloged in the system database directory.

iCallerAction
Input. Specifies action to be taken. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_ROLLFWD
Rollforward to the point in time specified by the piStopTime parameter. For database rollforward, the database is left in rollforward-pending state. For table space rollforward to a point in time, the table spaces are left in rollforward-in-progress state.

DB2ROLLFORWARD_STOP
End roll-forward recovery by rolling forward the database using available log files and then rolling it back. Uncommitted transactions are backed out and the rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_RFWD_STOP
Rollforward to the point in time specified by piStopTime, and end roll-forward recovery. The rollforward-pending state of the database or table spaces is turned off. A synonym for this value is DB2ROLLFORWARD_RFWD_COMPLETE.

DB2ROLLFORWARD_QUERY
Query values for nextarclog, firstarcdel, lastarcdel, and lastcommit. Return database status and a node number.

DB2ROLLFORWARD_PARM_CHECK
Validate parameters without performing the roll forward.

DB2ROLLFORWARD_CANCEL
Cancel the rollforward operation that is currently running. The database or table space are put in recovery pending state.

Note: This option cannot be used while the rollforward is actually running. It can be used if the rollforward is paused (that is, waiting for a STOP), or if a system failure occurred during the rollforward. It should be used with caution.

Rolling databases forward may require a load recovery using tape devices. The rollforward API will return with a warning message if user intervention on a device is required. The API can be called again with one of the following three caller actions:

DB2ROLLFORWARD_LOADREC_CONT
Continue using the device that generated the warning message (for example, when a new tape has been mounted).

DB2ROLLFORWARD_DEVICE_TERM
Stop using the device that generated the warning message (for example, when there are no more tapes).

DB2ROLLFORWARD_LOAD_REC_TERM
Terminate all devices being used by load recovery.

piStopTime
Input. A character string containing a time stamp in ISO format. Database recovery will stop when this time stamp is exceeded. Specify SQLUM_INFINITY_TIMESTAMP to roll forward as far as possible. May be NULL for DB2ROLLFORWARD_QUERY, DB2ROLLFORWARD_PARM_CHECK, and any of the load recovery (DB2ROLLFORWARD_LOADREC_xxx) caller actions.

piUserName
Input. A string containing the user name of the application. Can be NULL.

piPassword
Input. A string containing the password of the supplied user name (if any). Can be NULL.

piOverflowLogPath
Input. This parameter is used to specify an alternate log path to be used. In addition to the active log files, archived log files need to be moved (by the user) into the logpath before they can be used by this utility. This can be a problem if the database does not have sufficient space in the logpath. The overflow log path is provided for this reason. During roll-forward recovery, the required log files are searched, first in the logpath, and then in the overflow log path. The log files needed for table space roll-forward recovery can be brought into either the logpath or the overflow log path. If the caller does not specify an overflow log path, the default value is the logpath. In a partitioned database environment, the overflow log path must be a valid, fully qualified path; the default path is the default overflow log path for each node. In a single-partition database environment, the overflow log path can be relative if the server is local.

iNumChngLgOvrflw
Input. Partitioned database environments only. The number of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

piChngLogOvrflw
Input. Partitioned database environments only. A pointer to a structure containing the fully qualified names of changed overflow log paths. These new log paths override the default overflow log path for the specified database partition server only.

iConnectMode
Input. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_OFFLINE
Offline roll forward. This value must be specified for database roll-forward recovery.

DB2ROLLFORWARD_ONLINE
Online roll forward.

piTablespaceList
Input. A pointer to a structure containing the names of the table spaces to be rolled forward to the end-of-logs or to a specific point in time. If not specified, the table spaces needing rollforward will be selected.

For partitioned tables, point in time (PIT) roll-forward of a table space containing any piece of a partitioned table must also roll forward all of the other table spaces in which that table resides to the same point in time. Roll forward to the end of the logs for a single table space containing a piece of a partitioned table is still allowed.

If a partitioned table has any attached, detached or dropped data partitions, then PIT roll-forward must include all table spaces for these data partitions as well. To determine if a partitioned table has any attached, detached, or dropped data partitions, query the Status field of the SYSDATAPARTITIONS catalog table.

Because a partitioned table can reside in multiple table spaces, it is generally necessary to roll forward multiple table spaces. Data that is recovered via dropped table recovery is written to the export directory specified in the piExportDir parameter. It is possible to roll forward all table spaces in one command, or do repeated roll-forward operations for subsets of the table spaces involved. A warning will be written to the notify log if the db2Rollforward API did not specify the full set of the table spaces necessary to recover all the data for the table. A warning will be returned to the user with full details of all partitions not recovered on the command found in the administration notification log.

Allowing the roll forward of a subset of the table spaces makes it easier to deal with cases where there is more data to be recovered than can fit into a single export directory.

iAllNodeFlag
Input. Partitioned database environments only. Indicates whether the rollforward operation is to be applied to all database partition servers defined in db2nodes.cfg. Valid values are:

DB2_NODE_LIST
Apply to database partition servers in a list that is passed in piNodeList.

DB2_ALL_NODES
Apply to all database partition servers. This is the default value. The piNodeList parameter must be set to NULL, if this value is used.

DB2_ALL_EXCEPT
Apply to all database partition servers except those in a list that is passed in piNodeList.

DB2_CAT_NODE_ONLY
Apply to the catalog partition only. The piNodeList parameter must be set to NULL, if this value is used.

iNumNodes
Input. Specifies the number of database partition servers in the piNodeList array.

piNodeList
Input. A pointer to an array of database partition server numbers on which to perform the roll-forward recovery.

iNumNodeInfo
Input. Defines the size of the output parameter poNodeInfo, which must be large enough to hold status information from each database partition that is being rolled forward. In a single-partition database environment, this parameter should be set to 1. The value of this parameter should be the same as the number of database partition servers for which this API is being called.

piDroppedTblID
Input. A string containing the ID of the dropped table whose recovery is being attempted. For partitioned tables, the drop-table-id identifies the table as a whole, so that all data partitions of the table can be recovered in a single roll-forward command.

piExportDir
Input. The name of the directory into which the dropped table data will be exported.

iRollforwardFlags
Input. Specifies the rollforward flags. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2ROLLFORWARD_EMPTY_FLAG
No flags specified.

DB2ROLLFORWARD_LOCAL_TIME
Allows the user to rollforward to a point in time that is the user's local time rather than GMT time. This makes it easier for users to rollforward to a specific point in time on their local machines, and eliminates potential user errors due to the translation of local to GMT time.

DB2ROLLFORWARD_NO_RETRIEVE
Controls which log files to be rolled forward on the standby machine by allowing the user to disable the retrieval of archived logs. By controlling the log files to be rolled forward, one can ensure that the standby machine is X hours behind the production

machine, to prevent the user affecting both systems. This option is useful if the standby system does not have access to archive, for example, if TSM is the archive, it only allows the original machine to retrieve the files. It will also remove the possibility that the standby system would retrieve an incomplete log file while the production system is archiving a file and the standby system is retrieving the same file.

db2RfwdOutputStruct data structure parameters:

poApplicationId
Output. The application ID.

poNumReplies
Output. The number of replies received.

poNodeInfo
Output. Database partition reply information.

oRollforwardFlags
Output. Rollforward output flags. Valid values are:

DB2ROLLFORWARD_OUT_LOCAL_TIME
Indicates to user that the last committed transaction timestamp is displayed in local time rather than UTC. Local time is based on the server's local time, not on the client's. In a partitioned database environment, local time is based on the catalog partition's local time.

sqlurf_newlogpath data structure parameters:

nodenum
Input. The number of the database partition that this structure details.

pathlen
Input. The total length of the logpath field.

logpath
Input. A fully qualified path to be used for a specific node for the rollforward operation.

sqlu_tablespace_bkrst_list data structure parameters:

num_entry
Input. The number of structures contained in the list pointed to by the tablespace parameter.

tablespace
Input. A pointer to a list of sqlu_tablespace_entry structures.

sqlu_tablespace_entry data structure parameters:

reserve_len
Input. Specifies the length in bytes of the tablespace_entry parameter.

tablespace_entry
Input. The name of the table space to rollforward.

filler Filler used for proper alignment of data structure in memory.

sqlurf_info data structure parameters:

nodenum
Output. The number of the database partition that this structure contains information for.

state
Output. The current state of the database or tablespaces that were included in the rollfoward on a database partition.

nextarclog
Output. If the rollforward has completed, this field will be empty. If the rollforward has not yet completed, this will be the name of the next log file which will be processed for the rollforward.

firstarcdel
Output. The first log file replayed by the rollforward.

lastarcdel
Output. The last log file replayed by the rollforward.

lastcommit
Output. The time of the last committed transaction.

db2gRfwdInputStruct data structure specific parameters:

iDbAliasLen
Input. Specifies the length in bytes of the database alias.

iStopTimeLen
Input. Specifies the length in bytes of the stop time parameter. Set to zero if no stop time is provided.

iUserNameLen
Input. Specifies the length in bytes of the user name. Set to zero if no user name is provided.

iPasswordLen
Input. Specifies the length in bytes of the password. Set to zero if no password is provided.

iOverflowLogPathLen
Input. Specifies the length in bytes of the overflow log path. Set to zero if no overflow log path is provided.

iDroppedTblIDLen
Input. Specifies the length in bytes of the dropped table ID (piDroppedTblID parameter). Set to zero if no dropped table ID is provided.

iExportDirLen
Input. Specifies the length in bytes of the dropped table export directory (piExportDir parameter). Set to zero if no dropped table export directory is provided.

Usage notes:

The database manager uses the information stored in the archived and the active log files to reconstruct the transactions performed on the database since its last backup.

The action performed when this API is called depends on the rollforward_pending flag of the database prior to the call. This can be queried using db2CfgGet - Get Configuration Parameters. The rollforward_pending flag is set to DATABASE if the database is in roll-forward pending state. It is set to TABLESPACE if one or more table spaces are in SQLB_ROLLFORWARD_PENDING or SQLB_ROLLFORWARD_IN_PROGRESS state. The rollforward_pending flag is set to NO if neither the database nor any of the table spaces needs to be rolled forward.

If the database is in roll-forward pending state when this API is called, the database will be rolled forward. Table spaces are returned to normal state after a successful database roll-forward, unless an abnormal state causes one or more table spaces to go offline. If the rollforward_pending flag is set to TABLESPACE, only those table spaces that are in roll-forward pending state, or those table spaces requested by name, will be rolled forward.

Note: If table space rollforward terminates abnormally, table spaces that were being rolled forward will be put in SQLB_ROLLFORWARD_IN_PROGRESS state. In the next invocation of ROLLFORWARD DATABASE, only those table spaces in SQLB_ROLLFORWARD_IN_PROGRESS state will be processed. If the set of selected table space names does not include all table spaces that are in SQLB_ROLLFORWARD_IN_PROGRESS state, the table spaces that are not required will be put into SQLB_RESTORE_PENDING state.

If the database is not in roll-forward pending state and no point in time is specified, any table spaces that are in rollforward-in-progress state will be rolled forward to the end of logs. If no table spaces are in rollforward-in-progress state, any table spaces that are in rollforward pending state will be rolled forward to the end of logs.

This API reads the log files, beginning with the log file that is matched with the backup image. The name of this log file can be determined by calling this API with a caller action of DB2ROLLFORWARD_QUERY before rolling forward any log files.

The transactions contained in the log files are reapplied to the database. The log is processed as far forward in time as information is available, or until the time specified by the stop time parameter.

Recovery stops when any one of the following events occurs:
- No more log files are found
- A time stamp in the log file exceeds the completion time stamp specified by the stop time parameter
- An error occurs while reading the log file.

Some transactions might not be recovered. The value returned in lascommit indicates the time stamp of the last committed transaction that was applied to the database.

If the need for database recovery was caused by application or human error, the user may want to provide a time stamp value in piStopTime, indicating that recovery should be stopped before the time of the error. This applies only to full database roll-forward recovery, and to table space rollforward to a point in time. It also permits recovery to be stopped before a log read error occurs, determined during an earlier failed attempt to recover.

When the rollforward_recovery flag is set to DATABASE, the database is not available for use until roll-forward recovery is terminated. Termination is accomplished by calling the API with a caller action of DB2ROLLFORWARD_STOP or DB2ROLLFORWARD_RFWRD_STOP to bring the database out of roll-forward pending state. If the rollforward_recovery flag is TABLESPACE, the database is available for use. However, the table spaces in SQLB_ROLLFORWARD_PENDING and SQLB_ROLLFORWARD_IN_PROGRESS states will not be available until the API is called to perform table space roll-forward recovery. If rolling forward table spaces to a point in time, the table spaces are placed in backup pending state after a successful rollforward.

When the RollforwardFlags option is set to DB2ROLLFORWARD_LOCAL_TIME, all messages returned to the user will also be in local time. All times are converted on the server, and on the catalog partition, if it is a partitioned database environment. The timestamp string is converted to GMT on the server, so the time is local to the server's time zone, not the client's. If the client is in one time zone and the server in another, the server's local time should be used. This is different from the local time option from the Control Center, which is local to the client. If the timestamp string is close to the time change of the clock due to daylight savings, it is important to know if the stop time is before or after the clock change, and specify it correctly.

Related tasks:
- "Using rollforward" in *Data Recovery and High Availability Guide and Reference*

Related reference:
- "SQLCA data structure" in *Administrative API Reference*
- "ROLLFORWARD DATABASE " on page 606
- "db2Backup - Back up a database or table space" on page 665
- "db2Restore - Restore a database or table space" on page 756
- "db2Recover - Restore and roll forward a database" on page 743

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "dbrecov.sqC -- How to recover a database (C++)"

db2SetWriteForDB - Suspend or resume I/O writes for database

Sets the database to be I/O write suspended, or resumes I/O writes to disk. I/O writes must be suspended for a database before a split mirror can be taken. To avoid potential problems, keep the same connection to do the write suspension and resumption.

Scope:

This API only affects the database partition on which it is executed.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint

Required connection:

Database

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2SetWriteForDB (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef struct db2SetWriteDbStruct
{
   db2int32 iOption;
   char *piTablespaceNames;
} db2SetWriteDbStruct;
```

db2SetWriteForDB API parameters:

versionNumber
Input. Specifies the version and release level of the structure passed as the second parameter pParmStruct.

pParmStruct
Input. A pointer to the db2SetWriteDbStruct structure.

pSqlca
Output. A pointer to the sqlca structure.

db2SetWriteDbStruct data structure parameters:

iOption
Input. Specifies the action. Valid values are:

- DB2_DB_SUSPEND_WRITE
Suspends I/O write to disk.

- DB2_DB_RESUME_WRITE
Resumes I/O write to disk.

piTablespaceNames
Input. Reserved for future use.

Related tasks:

- "Using a split mirror as a backup image" in *Data Recovery and High Availability Guide and Reference*
- "Using a split mirror as a standby database" in *Data Recovery and High Availability Guide and Reference*
- "Using a split mirror to clone a database" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "SET WRITE " on page 616
- "SQLCA data structure" in *Administrative API Reference*

sqlabndx - Bind application program to create a package

Invokes the bind utility, which prepares SQL statements stored in the bind file generated by the precompiler, and creates a package that is stored in the database.

Scope:

This API can be called from any database partition server in db2nodes.cfg. It updates the database catalogs on the catalog partition. Its effects are visible to all database partition servers.

Authorization:

One of the following:
- sysadm or dbadm authority
- BINDADD privilege if a package does not exist and one of:
- IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
- CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has sysadm authority, but not explicit privileges to complete the bind, the database manager grants explicit dbadm authority automatically.

Required connection:

Database

API include file:

```
sql.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlabndx (
        _SQLOLDCHAR * pBindFileName,
        _SQLOLDCHAR * pMsgFileName,
        struct sqlopt * pBindOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgbndx (
        unsigned short MsgFileNameLen,
        unsigned short BindFileNameLen,
        struct sqlca * pSqlca,
        struct sqlopt * pBindOptions,
        _SQLOLDCHAR * pMsgFileName,
        _SQLOLDCHAR * pBindFileName);
```

sqlabndx API parameters:

pBindFileName

Input. A string containing the name of the bind file, or the name of a file containing a list of bind file names. The bind file names must contain the extension .bnd. A path for these files can be specified.

Precede the name of a bind list file with the at sign (@). For example, a fully qualified bind list file name might be:

```
/u/user1/bnd/@all.lst
```

The bind list file should contain one or more bind file names, and must have the extension .lst.

Precede all but the first bind file name with a plus symbol (+). The bind file names may be on one or more lines. For example, the bind list file all.lst might contain:

```
mybind1.bnd+mybind2.bnd+
mybind3.bnd+
mybind4.bnd
```

Path specifications on bind file names in the list file can be used. If no path is specified, the database manager takes path information from the bind list file.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pBindOptions
Input. A structure used to pass bind options to the API. For more information about this structure, see SQLOPT.

pSqlca
Output. A pointer to the sqlca structure.

sqlgbndx API-specific parameters:

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

BindFileNameLen
Input. Length in bytes of the pBindFileName parameter.

Usage notes:

Binding can be done as part of the precompile process for an application program source file, or as a separate step at a later time. Use BIND when binding is performed as a separate process.

The name used to create the package is stored in the bind file, and is based on the source file name from which it was generated (existing paths or extensions are discarded). For example, a precompiled source file called myapp.sqc generates a default bind file called myapp.bnd and a default package name of MYAPP. (However, the bind file name and the package name can be overridden at precompile time by using the SQL_BIND_OPT and the SQL_PKG_OPT options of sqlaprep.)

BIND executes under the transaction that the user has started. After performing the bind, BIND issues a COMMIT (if bind is successful) or a ROLLBACK (if bind is unsuccessful) operation to terminate the current transaction and start another one.

Binding halts if a fatal error or more than 100 errors occur. If a fatal error occurs during binding, BIND stops binding, attempts to close all files, and discards the package.

Binding application programs have prerequisite requirements and restrictions beyond the scope of this manual. For example, an application cannot be bound from a Version 8 client to a Version 8 server, and then executed against a Version 7 server.

The Bind option types and values are defined in sql.h.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:
- “Calling DB2 APIs in REXX” in *Administrative API Reference*

Related reference:
- “sqlarbnd - Rebind package” on page 833
- “sqlaprep - Precompile application program” on page 831
- “SQLCA data structure” in *Administrative API Reference*
- “sqlchar data structure” in *Administrative API Reference*
- “sqlopt data structure” in *Administrative API Reference*
- “BIND” on page 441

Related samples:
- “dbsample.sqc -- Creates a sample database (C)”
- “dbpkg.sqc -- How to work with packages (C)”
- “dbpkg.sqC -- How to work with packages (C++)”

sqlbftpq - Fetch the query data for rows in a table space

Fetches a specified number of rows of table space query data, each row consisting of data for a table space.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint

- dbadm
- load

Required connection:

Database

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlbftpq (
   struct sqlca * pSqlca,
   sqluint32 MaxTablespaces,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 * pNumTablespaces);

SQL_API_RC SQL_API_FN
  sqlgftpq (
   struct sqlca * pSqlca,
   sqluint32 MaxTablespaces,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 * pNumTablespaces);
```

sqlbftpq API parameters:

pSqlca
Output. A pointer to the sqlca structure.

MaxTablespaces
Input. The maximum number of rows of data that the user allocated output area (pointed to by pTablespaceData) can hold.

pTablespaceData
Input and output. Pointer to the output area, a structure for query data. For more information about this structure, see SQLB-TBSPQRY-DATA. The caller of this API must:

- Allocate space for MaxTablespaces of these structures
- Initialize the structures
- Set TBSPQVER in the first structure to SQLB_TBSPQRY_DATA_ID
- Set pTablespaceData to point to this space. The API will use this space to return the table space data.

pNumTablespaces
Output. Number of rows of output returned.

Usage notes:

The user is responsible for allocating and freeing the memory pointed to by the pTablespaceData parameter. This API can only be used after a successful sqlbotsq call. It can be invoked repeatedly to fetch the list generated by sqlbotsq.

Related reference:

- “sqlbctsq API - Close a table space query” in *Administrative API Reference*
- “sqlbgtss API - Get table space usage statistics” in *Administrative API Reference*
- “sqlbmtsq - Get the query data for all table spaces” on page 783

- "sqlbotsq API - Open a table space query" in *Administrative API Reference*
- "sqlbstpq - Get information about a single table space" on page 786
- "SQLB_TBSPQRY_DATA data structure" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

Related samples:
- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbmtsq - Get the query data for all table spaces

Provides a one-call interface to the table space query data. The query data for all table spaces in the database is returned in an array.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint
- dbadm
- load

Required connection:

Database

API include file:

`sqlutil.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlbmtsq (
   struct sqlca * pSqlca,
   sqluint32 * pNumTablespaces,
   struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
   sqluint32 reserved1,
   sqluint32 reserved2);

SQL_API_RC SQL_API_FN
  sqlgmtsq (
   struct sqlca * pSqlca,
   sqluint32 * pNumTablespaces,
   struct SQLB_TBSPQRY_DATA *** pppTablespaceData,
   sqluint32 reserved1,
   sqluint32 reserved2);
```

sqlbmtsq API parameters:

pSqlca
Output. A pointer to the sqlca structure.

pNumTablespaces
Output. The total number of table spaces in the connected database.

pppTablespaceData
Output. The caller supplies the API with the address of a pointer. The space for the table space query data is allocated by the API, and a pointer to that space is returned to the caller. On return from the call, the pointer points to an array of SQLB_TBSPQRY_DATA pointers to the complete set of table space query data.

reserved1
Input. Always SQLB_RESERVED1.

reserved2
Input. Always SQLB_RESERVED2.

Usage notes:

This API uses the lower level services, namely:
- sqlbotsq
- sqlbftpq
- sqlbctsq

to get all of the table space query data at once.

If sufficient memory is available, this function returns the number of table spaces, and a pointer to the memory location of the table space query data. It is the user's responsibility to free this memory with a call to sqlefmem.

If sufficient memory is not available, this function simply returns the number of table spaces, and no memory is allocated. If this should happen, use sqlbotsq, sqlbftpq, and sqlbctsq, to fetch less than the whole list at once.

Related reference:
- "LIST TABLESPACES " on page 537
- "sqlbctsq API - Close a table space query" in *Administrative API Reference*
- "sqlbftpq - Fetch the query data for rows in a table space" on page 781
- "sqlbgtss API - Get table space usage statistics" in *Administrative API Reference*
- "sqlbotsq API - Open a table space query" in *Administrative API Reference*
- "sqlbstpq - Get information about a single table space" on page 786
- "sqlefmem API - Free the memory allocated by the sqlbtcq and sqlbmtsq API" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "sqlbtcq API - Get the query data for all table space containers" in *Administrative API Reference*

Related samples:
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "dbrecov.sqc -- How to recover a database (C)"

- "tsinfo.sqC -- How to get information at the table space level (C++)"
- "dbrecov.sqC -- How to recover a database (C++)"

sqlbotcq - Open a table space container query

Prepares for a table space container query operation, and returns the number of containers currently in the table space.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint
- dbadm

Required connection:

Database

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlbotcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers);

SQL_API_RC SQL_API_FN
  sqlgotcq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   sqluint32 * pNumContainers);
```

sqlbotcq API parameters:

pSqlca
Output. A pointer to the sqlca structure.

TablespaceId
Input. ID of the table space for which container data is desired. If the special identifier SQLB_ALL_TABLESPACES (in sqlutil.h) is specified, a complete list of containers for the entire database is produced.

pNumContainers
Output. The number of containers in the specified table space.

Usage notes:

This API is normally followed by one or more calls to sqlbftcq, and then by one call to sqlbctcq.

An application can use the following APIs to fetch information about containers in use by table spaces:
- sqlbtcq

Fetches a complete list of container information. The API allocates the space required to hold the information for all the containers, and returns a pointer to this information. Use this API to scan the list of containers for specific information. Using this API is identical to calling the three APIs below (sqlbotcq, sqlbftcq, sqlbctcq), except that this API automatically allocates the memory for the output information. A call to this API must be followed by a call to sqlefmem to free the memory.

- sqlbotcq
- sqlbftcq
- sqlbctcq

  These three APIs function like an SQL cursor, in that they use the OPEN/FETCH/CLOSE paradigm. The caller must provide the output area for the fetch. Unlike an SQL cursor, only one table space container query can be active at a time. Use this set of APIs to scan the list of table space containers for specific information. These APIs allows the user to control the memory requirements of an application (compared with sqlbtcq).

When sqlbotcq is called, a snapshot of the current container information is formed in the agent servicing the application. If the application issues a second table space container query call (sqlbtcq or sqlbotcq), this snapshot is replaced with refreshed information.

No locking is performed, so the information in the buffer may not reflect changes made by another application after the snapshot was generated. The information is not part of a transaction.

There is one snapshot buffer for table space queries and another for table space container queries. These buffers are independent of one another.

Related reference:

- "sqlbctcq API - Close a table space container query" in *Administrative API Reference*
- "sqlbftcq API - Fetch the query data for rows in a table space container" in *Administrative API Reference*
- "sqlbstsc API - Set table space containers" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "sqlbtcq API - Get the query data for all table space containers" in *Administrative API Reference*

Related samples:

- "tabscont.sqb -- How to get tablespace container information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqlbstpq - Get information about a single table space

Retrieves information about a single currently defined table space.

Scope:

In a partitioned database environment, only the table spaces on the current database partition are listed.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint
- dbadm
- load

Required connection:

Database

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlbstpq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 reserved);

SQL_API_RC SQL_API_FN
  sqlgstpq (
   struct sqlca * pSqlca,
   sqluint32 TablespaceId,
   struct SQLB_TBSPQRY_DATA * pTablespaceData,
   sqluint32 reserved);
```

sqlbstpq API parameters:

pSqlca
: Output. A pointer to the sqlca structure.

TablespaceId
: Input. Identifier for the table space which is to be queried.

pTablespaceData
: Input and output. Pointer to a user-supplied SQLB_TBSPQRY_DATA structure where the table space information will be placed upon return. The caller of this API must initialize the structure and set TBSPQVER to SQLB_TBSPQRY_DATA_ID (in sqlutil).

reserved
: Input. Always SQLB_RESERVED1.

Usage notes:

This API retrieves information about a single table space if the table space identifier to be queried is known. This API provides an alternative to the more expensive OPEN TABLESPACE QUERY, FETCH, and CLOSE combination of APIs, which must be used to scan for the desired table space when the table space identifier is not known in advance. The table space IDs can be found in the system catalogs. No agent snapshot is taken; since there is only one entry to return, it is returned directly.

Related reference:
- "sqlbctsq API - Close a table space query" in *Administrative API Reference*
- "sqlbftpq - Fetch the query data for rows in a table space" on page 781
- "sqlbgtss API - Get table space usage statistics" in *Administrative API Reference*
- "sqlbmtsq - Get the query data for all table spaces" on page 783
- "sqlbotsq API - Open a table space query" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

Related samples:
- "tabspace.sqb -- How to get tablespace information (IBM COBOL)"
- "tspace.sqb -- How to copy and free memory in a tablespace (IBM COBOL)"
- "tsinfo.sqc -- How to get information at the table space level (C)"
- "tsinfo.sqC -- How to get information at the table space level (C++)"

sqle_activate_db - Activate database

Activates the specified database and starts up all necessary database services, so that the database is available for connection and use by any application.

Scope:

This API activates the specified database on all database partition servers. If one or more of these database partition servers encounters an error during activation of the database, a warning is returned. The database remains activated on all database partition servers on which the API has succeeded.

Note: If it is the coordinator partition or the catalog partition that encounters the error, the API returns a negative sqlcode, and the database will not be activated on any database partition server.

Authorization:

One of the following:
- sysadm
- sysctrl
- sysmaint

Required connection:

None. Applications invoking ACTIVATE DATABASE cannot have any existing database connections.

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqle_activate_db (
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlg_activate_db (
        unsigned short DbAliasLen,
        unsigned short UserNameLen,
        unsigned short PasswordLen,
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);
```

sqle_activate_db API parameters:

pDbAlias
Input. Pointer to the database alias name.

pUserName
Input. Pointer to the user ID starting the database. Can be NULL.

pPassword
Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlg_activate_db API-specific parameters:

DbAliasLen
Input. A 2-byte unsigned integer representing the length of the database alias name in bytes.

UserNameLen
Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

PasswordLen
Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

Usage notes:

If a database has not been started, and a DB2 CONNECT TO (or an implicit connect) is encountered in an application, the application must wait while the database manager starts up the required database. In such cases, this first application spends time on database initialization before it can do any work. However, once the first application has started a database, other applications can simply connect and use it.

Database administrators can use ACTIVATE DATABASE to start up selected databases. This eliminates any application time spent on database initialization.

Databases initialized by ACTIVATE DATABASE can only be shut down by sqle_deactivate_db, or by db2InstanceStop. To obtain a list of activated databases, call db2GetSnapshot.

If a database was started by a DB2 CONNECT TO (or an implicit connect) and subsequently an ACTIVATE DATABASE is issued for that same database, then DEACTIVATE DATABASE must be used to shut down that database.

ACTIVATE DATABASE behaves in a similar manner to a DB2 CONNECT TO (or an implicit connect) when working with a database requiring a restart (for example, database in an inconsistent state). The database will be restarted before it can be initialized by ACTIVATE DATABASE.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:
- "Calling DB2 APIs in REXX" in *Administrative API Reference*
- "Quick-start tips for performance tuning" in *Performance Guide*

Related tasks:
- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:
- "db2GetSnapshot API - Get a snapshot of the database manager operational status" in *Administrative API Reference*
- "sqle_deactivate_db - Deactivate database" on page 790
- "SQLCA data structure" in *Administrative API Reference*
- "ACTIVATE DATABASE command" in *Command Reference*

sqle_deactivate_db - Deactivate database

Stops the specified database.

Scope:

In a partitioned database environment, this API deactivates the specified database on all database partition servers. If one or more of these database partition servers encounters an error, a warning is returned. The database will be successfully deactivated on some database partition servers, but may remain activated on the database partition servers encountering the error.

Note: If it is the coordinator partition or the catalog partition that encounters the error, the API returns a negative sqlcode, and the database will not be reactivated on any database partition server on which it was deactivated.

Authorization:

One of the following:
- sysadm
- sysctrl

- sysmaint

Required connection:

None. Applications invoking DEACTIVATE DATABASE cannot have any existing database connections.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqle_deactivate_db (
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlg_deactivate_db (
        unsigned short DbAliasLen,
        unsigned short UserNameLen,
        unsigned short PasswordLen,
        char * pDbAlias,
        char * pUserName,
        char * pPassword,
        void * pReserved,
        struct sqlca * pSqlca);
```

sqle_deactivate_db API parameters:

pDbAlias
: Input. Pointer to the database alias name.

pUserName
: Input. Pointer to the user ID stopping the database. Can be NULL.

pPassword
: Input. Pointer to the password for the user name. Can be NULL, but must be specified if a user name is specified.

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the sqlca structure.

sqlg_deactivate_db API-specific parameters:

DbAliasLen
: Input. A 2-byte unsigned integer representing the length of the database alias name in bytes.

UserNameLen
: Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

PasswordLen
: Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

Usage notes:

Databases initialized by ACTIVATE DATABASE can only be shut down by DEACTIVATE DATABASE. db2InstanceStop automatically stops all activated databases before stopping the database manager. If a database was initialized by ACTIVATE DATABASE, the last DB2 CONNECT RESET statement (counter equal 0) will not shut down the database; DEACTIVATE DATABASE must be used.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- "Calling DB2 APIs in REXX" in *Administrative API Reference*
- "Quick-start tips for performance tuning" in *Performance Guide*

Related tasks:

- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:

- "sqle_activate_db - Activate database" on page 788
- "SQLCA data structure" in *Administrative API Reference*
- "DEACTIVATE DATABASE command" in *Command Reference*

sqleaddn - Add a database partition server to the partitioned database environment

Adds a new database partition server to the partitioned database environment. This API creates database partitions for all databases currently defined in the instance on the new database partition server. The user can specify the source database partition server for any system temporary table spaces to be created with the databases, or specify that no system temporary table spaces are to be created. The API must be issued from the database partition server that is being added, and can only be issued on a database partition server.

Scope:

This API only affects the database partition server on which it is executed.

Authorization:

One of the following:

- sysadm
- sysctrl

Required connection:

None

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqleaddn (
        void * pAddNodeOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgaddn (
        unsigned short addnOptionsLen,
        struct sqlca * pSqlca,
        void * pAddNodeOptions);
```

sqleaddn API parameters:

pAddNodeOptions
Input. A pointer to the optional sqle_addn_options structure. This structure is used to specify the source database partition server, if any, of the system temporary table space definitions for all database partitions created during the add node operation. If not specified (that is, a NULL pointer is specified), the system temporary table space definitions will be the same as those for the catalog partition.

pSqlca
Output. A pointer to the sqlca structure.

sqlgaddn API-specific parameters:

addnOptionsLen
Input. A 2-byte unsigned integer representing the length of the optional sqle_addn_options structure in bytes.

Usage notes:

Before adding a new database partition server, ensure that there is sufficient storage for the containers that must be created for all existing databases on the system.

The add node operation creates an empty database partition on the new database partition server for every database that exists in the instance. The configuration parameters for the new database partitions are set to the default value.

If an add node operation fails while creating a database partition locally, it enters a clean-up phase, in which it locally drops all databases that have been created. This means that the database partitions are removed only from the database partition server being added (that is, the local database partition server). Existing database partitions remain unaffected on all other database partition servers. If this fails, no further clean up is done, and an error is returned.

The database partitions on the new database partition server cannot be used to contain user data until after the ALTER DATABASE PARTITION GROUP statement has been used to add the database partition server to a database partition group.

This API will fail if a create database or a drop database operation is in progress. The API can be called again once the operation has completed.

This API will fail if at any time in a database in the system a user table with an XML column had been, successfully or not, created or an XSR object had been, successfully or not, registered.

To determine whether or not a database is enabled for automatic storage, the sqleaddn API has to communicate with the catalog partition for each of the databases in the instance. If automatic storage is enabled then the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the sqleaddn API may have to communicate with another database partition server in the partitioned database environment in order to retrieve the table space definitions. The start_stop_time database manager configuration parameter is used to specify the time, in minutes, by which the other database partition server must respond with the automatic storage and table space definitions. If this time is exceeded, the API fails. Increase the value of start_stop_time, and call the API again.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:
- "Calling DB2 APIs in REXX" in *Administrative API Reference*

Related tasks:
- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:
- "sqlecrea - Create database" on page 800
- "sqledrpn - Check whether a database partition server can be dropped" on page 810
- "SQLCA data structure" in *Administrative API Reference*
- "sqle_addn_options data structure" in *Administrative API Reference*
- "ALTER DATABASE PARTITION GROUP " on page 845

sqlecadb - Catalog a database in the system database directory

Stores database location information in the system database directory. The database can be located either on the local workstation or on a remote node.

Scope:

This API affects the system database directory. In a partitioned database environment, when cataloging a local database into the system database directory, this API must be called from a database partition server where the database resides.

Authorization:

One of the following:
- sysadm
- sysctrl

Required connection:

None

API include file:

sqlenv.h

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlecadb (
        _SQLOLDCHAR * pDbName,
        _SQLOLDCHAR * pDbAlias,
        unsigned char Type,
        _SQLOLDCHAR * pNodeName,
        _SQLOLDCHAR * pPath,
        _SQLOLDCHAR * pComment,
        unsigned short Authentication,
        _SQLOLDCHAR * pPrincipal,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcadb (
        unsigned short PrinLen,
        unsigned short CommentLen,
        unsigned short PathLen,
        unsigned short NodeNameLen,
        unsigned short DbAliasLen,
        unsigned short DbNameLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pPrinName,
        unsigned short Authentication,
        _SQLOLDCHAR * pComment,
        _SQLOLDCHAR * pPath,
        _SQLOLDCHAR * pNodeName,
        unsigned char Type,
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pDbName);
```

sqlecadb API parameters:

pDbName
Input. A string containing the database name.

pDbAlias
Input. A string containing an alias for the database.

Type Input. A single character that designates whether the database is indirect, remote, or is cataloged via DCE. Valid values (defined in sqlenv) are:

SQL_INDIRECT
Specifies that the database resides at this instance.

SQL_REMOTE
Specifies that the database resides at another instance.

SQL_DCE
Specifies that the database is cataloged via DCE.

pNodeName
Input. A string containing the name of the node where the database is located. May be NULL.

Note: If neither pPath nor pNodeName is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter dftdbpath.

pPath

Input. A string which, on UNIX based systems, specifies the name of the path on which the database being cataloged resides. Maximum length is 215 characters.

On the Windows operating system, this string specifies the letter of the drive on which the database being cataloged resides.

If a NULL pointer is provided, the default database path is assumed to be that specified by the database manager configuration parameter *dftdbpath*.

pComment
Input. A string containing an optional description of the database. A null string indicates no comment. The maximum length of a comment string is 30 characters.

Authentication

Input. Contains the authentication type specified for the database. Authentication is a process that verifies that the user is who he/she claims to be. Access to database objects depends on the user's authentication. Valid values (from sqlenv) are:

SQL_AUTHENTICATION_SERVER
Specifies that authentication takes place on the node containing the target database.

SQL_AUTHENTICATION_CLIENT
Specifies that authentication takes place on the node where the application is invoked.

SQL_AUTHENTICATION_KERBEROS
Specifies that authentication takes place using Kerberos Security Mechanism.

SQL_AUTHENTICATION_NOT_SPECIFIED
Authentication not specified.

SQL_AUTHENTICATION_SVR_ENCRYPT
Specifies that authentication takes place on the node containing the target database, and that the authentication password is to be encrypted.

SQL_AUTHENTICATION_DATAENC
Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption.

SQL_AUTHENTICATION_GSSPLUGIN
Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

This parameter can be set to SQL_AUTHENTICATION_NOT_SPECIFIED, except when cataloging a database that resides on a DB2 Version 1 server.

Specifying the authentication type in the database catalog results in a performance improvement during a connect.

pPrincipal
Input. A string containing the principal name of the DB2 server on which the database resides. This value should only be specified when authentication is SQL_AUTHENTICATION_KERBEROS.

pSqlca
Output. A pointer to the sqlca structure.

sqlgcadb API-specific parameters:

PrinLen
: Input. A 2-byte unsigned integer representing the length in bytes of the principal name. Set to zero if no principal is provided. This value should be nonzero only when authentication is specified as SQL_AUTHENTICATION_KERBEROS.

CommentLen
: Input. A 2-byte unsigned integer representing the length in bytes of the comment. Set to zero if no comment is provided.

PathLen
: Input. A 2-byte unsigned integer representing the length in bytes of the path of the local database directory. Set to zero if no path is provided.

NodeNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the node name. Set to zero if no node name is provided.

DbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

DbNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database name.

pPrinName
: Input. A string containing the principal name of the DB2 server on which the database resides. This value should only be specified when authentication is SQL_AUTHENTICATION_KERBEROS.

Usage notes:

Use CATALOG DATABASE to catalog databases located on local or remote nodes, recatalog databases that were uncataloged previously, or maintain multiple aliases for one database (regardless of database location).

DB2 automatically catalogs databases when they are created. It catalogs an entry for the database in the local database directory, and another entry in the system database directory. If the database is created from a remote client (or a client which is executing from a different instance on the same machine), an entry is also made in the system database directory at the client instance.

Databases created at the current instance (as defined by the value of the DB2INSTANCE environment variable) are cataloged as indirect. Databases created at other instances are cataloged as remote (even if they physically reside on the same machine).

CATALOG DATABASE automatically creates a system database directory if one does not exist. The system database directory is stored on the path that contains the database manager instance that is being used. The system database directory is maintained outside of the database. Each entry in the directory contains:

- Alias
- Authentication type
- Comment
- Database

- Entry type
- Local database directory (when cataloging a local database)
- Node name (when cataloging a remote database)
- Release information

If a database is cataloged with the type parameter set to SQL_INDIRECT, the value of the authentication parameter provided will be ignored, and the authentication in the directory will be set to SQL_AUTHENTICATION_NOT_SPECIFIED.

If directory caching is enabled, database, node, and DCS directory files are cached in memory. An application's directory cache is created during its first directory lookup. Since the cache is only refreshed when the application modifies any of the directory files, directory changes made by other applications may not be effective until the application has restarted. To refresh DB2's shared cache (server only), stop (db2stop) and then restart (db2start) the database manager. To refresh the directory cache for another application, stop and then restart that application.

REXX API syntax:

```
CATALOG DATABASE dbname [AS alias] [ON path|AT NODE nodename]
[AUTHENTICATION authentication] [WITH "comment"]
CATALOG GLOBAL DATABASE db_global_name AS alias
USING DIRECTORY {DCE} [WITH "comment"]
```

REXX API parameters:

dbname
Name of the database to be cataloged.

alias Alternate name for the database. If an alias is not specified, the database name is used as the alias.

path Path on which the database being cataloged resides.

nodename
Name of the remote workstation where the database being cataloged resides.

Note: If neither path nor nodename is specified, the database is assumed to be local, and the location of the database is assumed to be that specified in the database manager configuration parameter dftdbpath.

authentication
Place where authentication is to be done. Valid values are:

SERVER
Authentication occurs at the node containing the target database. This is the default.

CLIENT
Authentication occurs at the node where the application is invoked.

KERBEROS
Specifies that authentication takes place using Kerberos Security Mechanism.

NOT_SPECIFIED
Authentication not specified.

SVR_ENCRYPT
Specifies that authentication takes place on the node containing the target database, and that the authentication password is to be encrypted.

DATAENC
Specifies that authentication takes place on the node containing the target database, and that connections must use data encryption.

GSSPLUGIN
Specifies that authentication takes place using an external GSS API-based plug-in security mechanism.

comment
Describes the database or the database entry in the system database directory. The maximum length of a comment string is 30 characters. A carriage return or a line feed character is not permitted. The comment text must be enclosed by double quotation marks.

db_global_name
The fully qualified name that uniquely identifies the database in the DCE name space.

DCE The global directory service being used.

REXX examples:

```
call SQLDBS 'CATALOG GLOBAL DATABASE /.../cell1/subsys/database/DB3
AS dbtest USING DIRECTORY DCE WITH "Sample Database"'
```

Related tasks:

- "Cataloging a database" in *Administration Guide: Implementation*

Related reference:

- "CATALOG DATABASE " on page 458
- "db2DbDirCloseScan API - End a system or local database directory scan" in *Administrative API Reference*
- "db2DbDirOpenScan API - Start a system or local database directory scan" in *Administrative API Reference*
- "sqlecrea - Create database" on page 800
- "sqleuncd API - Uncatalog a database from the system database directory" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*
- "db2DbDirGetNextEntry API - Get the next system or local database directory entry" in *Administrative API Reference*

Related samples:

- "dbcat.cbl -- Catalog to and uncatalog from a database (IBM COBOL)"
- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqlecrea - Create database

Initializes a new database with an optional user-defined collating sequence, creates the three initial table spaces, creates the system tables, and allocates the recovery log. To use XML features, you must set the `SQLDBCODESET` field of the `sqledbterritoryinfo` data structure to `UTF-8`.

Scope:

In a partitioned database environment, this API affects all database partition servers that are listed in the db2nodes.cfg file.

The database partition server from which this API is called becomes the catalog partition for the new database.

Authorization:

One of the following:
- sysadm
- sysctrl

Required connection:

Instance. To create a database at another (remote) node, it is necessary to first attach to that node. A database connection is temporarily established by this API during processing.

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlecrea (
        char * pDbName,
        char * pLocalDbAlias,
        char * pPath,
        struct sqledbdesc * pDbDescriptor,
        SQLEDBTERRITORYINFO * pTerritoryInfo,
        char Reserved2,
        void * pDbDescriptorExt,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgcrea (
        unsigned short PathLen,
        unsigned short LocalDbAliasLen,
        unsigned short DbNameLen,
        struct sqlca * pSqlca,
        void * pReserved1,
        unsigned short Reserved2,
        SQLEDBTERRITORYINFO * pTerritoryInfo,
        struct sqledbdesc * pDbDescriptor,
        char * pPath,
        char * pLocalDbAlias,
        char * pDbName);
```

sqlecrea API parameters:

pDbName
Input. A string containing the database name. This is the database name that will be cataloged in the system database directory. Once the database has been successfully created in the server's system database directory, it is automatically cataloged in the system database directory with a database alias identical to the database name. Must not be NULL.

pLocalDbAlias
Input. A string containing the alias to be placed in the client's system database directory. Can be NULL. If no local alias is specified, the database name is the default.

pPath Input. On Linux and UNIX systems, specifies the path on which to create the database. If a path is not specified, the database is created on the default database path specified in the database manager configuration file (dftdbpath parameter). On the Windows operating system, specifies the letter of the drive on which to create the database. Can be NULL.

Note: For partitioned database environments, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the dftdbpath database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the $HOME directory of the instance owner). The path specified for this API in a partitioned database environment cannot be a relative path.

pDbDescriptor
Input. A pointer to the database description block that is used when creating the database. The database description block can be used by you to supply values that are permanently stored in the configuration file of the database.

The supplied values are a collating sequence, a database comment, or a table space definition. The supplied value can be NULL if you do not want to supply any values. For information about the values that can be supplied through this parameter, see the SQLEDBDESC data structure topic.

pTerritoryInfo
Input. A pointer to the sqledbterritoryinfo structure, containing the locale and the code set for the database. Can be NULL. In a future release of the DB2 database system, the default code set will be changed to UTF-8 when creating a database. If a particular code set and territory is needed for a database, the desired code set and territory should be specified via the sqledbterritoryinfo structure.

Reserved2
Input. Reserved for future use.

pDbDescriptorExt
Input. This parameter refers to an extended database description block (sqledbdescext) that is used when creating the database. The extended database description block controls automatic storage for a database, chooses a default page size for the database, and specifies values for new table space attributes that have been introduced. If set to null or zero, a default page size of 4 096 bytes is chosen for the database and automatic storage is enabled.

pSqlca
Output. A pointer to the sqlca structure.

sqlgcrea API-specific parameters:

PathLen
: Input. A 2-byte unsigned integer representing the length of the path in bytes. Set to zero if no path is provided.

LocalDbALiasLen
: Input. A 2-byte unsigned integer representing the length of the local database alias in bytes. Set to zero if no local alias is provided.

DbNameLen
: Input. A 2-byte unsigned integer representing the length of the database name in bytes.

Usage notes:

CREATE DATABASE:

- Creates a database in the specified subdirectory. In a partitioned database environment, creates the database on all database partition servers listed in db2nodes.cfg, and creates a $DB2INSTANCE/NODExxxx directory under the specified subdirectory at each database partition server, where xxxx represents the local database partition server number. In a single-partition environment, creates a $DB2INSTANCE/NODE0000 directory under the specified subdirectory.
- Creates the system catalog tables and recovery log.
- Catalogs the database in the following database directories:
  - server's local database directory on the path indicated by pPath or, if the path is not specified, the default database path defined in the database manager system configuration file. A local database directory resides on each file system that contains a database.
  - server's system database directory for the attached instance. The resulting directory entry will contain the database name and a database alias.

    If the API was called from a remote client, the client's system database directory is also updated with the database name and an alias.
- Creates a system or a local database directory if neither exists. If specified, the comment and code set values are placed in both directories.
- Stores the specified code set, territory, and collating sequence. A flag is set in the database configuration file if the collating sequence consists of unique weights, or if it is the identity sequence.
- Creates the schemata called SYSCAT, SYSFUN, SYSIBM, and SYSSTAT with SYSIBM as the owner. The database partition server on which this API is called becomes the catalog partition for the new database. Two database partition groups are created automatically: IBMDEFAULTGROUP and IBMCATGROUP.
- Binds the previously defined database manager bind files to the database (these are listed in db2ubind.lst). If one or more of these files do not bind successfully, sqlecrea returns a warning in the SQLCA, and provides information about the binds that failed. If a bind fails, the user can take corrective action and manually bind the failing file. The database is created in any case. A schema called NULLID is implicitly created when performing the binds with CREATEIN privilege granted to PUBLIC.
- Creates SYSCATSPACE, TEMPSPACE1, and USERSPACE1 table spaces. The SYSCATSPACE table space is only created on the catalog partition. All database partitions have the same table space definitions.
- Grants the following:

- DBADM authority, and CONNECT, CREATETAB, BINDADD, CREATE_NOT_FENCED, IMPLICIT_SCHEMA, and LOAD privileges to the database creator
- CONNECT, CREATETAB, BINDADD, and IMPLICIT_SCHEMA privileges to PUBLIC
- USE privilege on the USERSPACE1 table space to PUBLIC
- SELECT privilege on each system catalog to PUBLIC
- BIND and EXECUTE privilege to PUBLIC for each successfully bound utility
- EXECUTE WITH GRANT privilege to PUBLIC on all functions in the SYSFUN schema.
- EXECUTE privilege to PUBLIC on all procedures in SYSIBM schema.

With dbadm authority, one can grant these privileges to (and revoke them from) other users or PUBLIC. If another administrator with sysadm or dbadm authority over the database revokes these privileges, the database creator nevertheless retains them.

In a partitioned database environment, the database manager creates a subdirectory, $DB2INSTANCE/NODExxxx, under the specified or default path on all database partition servers. The xxxx is the node number as defined in the db2nodes.cfg file (that is, node 0 becomes NODE0000). Subdirectories SQL00001 through SQLnnnnn will reside on this path. This ensures that the database objects associated with different database partition servers are stored in different directories (even if the subdirectory $DB2INSTANCE under the specified or default path is shared by all database partition servers).

On Windows and AIX operating systems, the length of the code set name is limited to a maximum of 9 characters. For example, specify a code set name such as ISO885915 instead of ISO8859-15.

Execution of the CREATE DATABASE command will fail if the application is already connected to a database.

If the database description block structure is not set correctly, an error message is returned.

The most prominent value of the database description block must be set to the symbolic value SQLE_DBDESC_2 (defined in sqlenv). The following sample user-defined collating sequences are available in the host language include files:

sqle819a
: If the code page of the database is 819 (ISO Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).

sqle819b
: If the code page of the database is 819 (ISO Latin/1),this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

sqle850a
: If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 500 (EBCDIC International).

sqle850b
If the code page of the database is 850 (ASCII Latin/1), this sequence will cause sorting to be performed according to the host CCSID 037 (EBCDIC US English).

sqle932a
If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5035 (EBCDIC Japanese).

sqle932b
If the code page of the database is 932 (ASCII Japanese), this sequence will cause sorting to be performed according to the host CCSID 5026 (EBCDIC Japanese).

The collating sequence specified during CREATE DATABASE cannot be changed later, and all character comparisons in the database use the specified collating sequence. This affects the structure of indexes as well as the results of queries.

Use sqlecadb to define different alias names for the new database.

The Configuration Advisor is called by default during the database creation process unless specifically told not to do so.

REXX API syntax:

```
CREATE DATABASE dbname [ON path] [ALIAS dbalias]
[USING CODESET codeset TERRITORY territory]
[COLLATE USING {SYSTEM | IDENTITY | USER :udcs}]
[NUMSEGS numsegs] [DFT_EXTENT_SZ dft_extentsize]
[CATALOG TABLESPACE <tablespace_definition>]
[USER TABLESPACE <tablespace_definition>]
[TEMPORARY TABLESPACE <tablespace_definition>]
[WITH comment]


Where <tablespace_definition> stands for:
MANAGED BY {
SYSTEM USING :SMS_string |
DATABASE USING :DMS_string }
[ EXTENTSIZE number_of_pages ]
[ PREFETCHSIZE number_of_pages ]
[ OVERHEAD number_of_milliseconds ]
[ TRANSFERRATE number_of_milliseconds ]
```

REXX API parameters:

dbname
Name of the database.

dbalias
Alias of the database.

path Path on which to create the database. If a path is not specified, the database is created on the default database path specified in the database manager configuration file (dftdbpath configuration parameter).

Note: For partitioned database environments, a database should not be created in an NFS-mounted directory. If a path is not specified, ensure that the dftdbpath database manager configuration parameter is not set to an NFS-mounted path (for example, on UNIX based systems, it should not specify the $HOME directory of the instance owner). The path specified for this API in a partitioned database environment cannot be a relative path.

codeset
Code set to be used for data entered into the database.

territory
Territory code (locale) to be used for data entered into the database.

SYSTEM
For non-Unicode databases, this is the default option, with the collating sequence based on the database territory. For Unicode databases, this option is equivalent to the IDENTITY option.

IDENTITY
Identity collating sequence, in which strings are compared byte for byte. This is the default for Unicode databases.

USER udcs
The collating sequence is specified by the calling application in a host variable containing a 256-byte string defining the collating sequence.

numsegs
Number of directories (tablespace containers) that will be created and used to store the database table files for any default SMS table spaces.

dft_extentsize
Specifies the default extent size for table spaces in the database.

SMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, and where the table space data will be stored. In the following, XXX represents the host variable name. Note that each of the directory names cannot exceed 254 bytes in length.

XXX.0 Number of directories specified

XXX.1 First directory name for SMS table space

XXX.2 Second directory name for SMS table space

XXX.3 and so on.

DMS_string
A compound REXX host variable identifying one or more containers that will belong to the table space, where the table space data will be stored, container sizes (specified in a number of 4KB pages) and types (file or device). The specified devices (not files) must already exist. In the following, XXX represents the host variable name. Note that each of the container names cannot exceed 254 bytes in length.

XXX.0 Number of strings in the REXX host variable (number of first level elements)

XXX.1.1
Type of the first container (file or device)

XXX.1.2
First file name or device name

XXX.1.3
Size (in pages) of the first container

XXX.2.1
Type of the second container (file or device)

XXX.2.2
Second file name or device name

XXX.2.3
Size (in pages) of the second container

XXX.3.1
and so on.

EXTENTSIZE number_of_pages
Number of 4KB pages that will be written to a container before skipping to the next container.

PREFETCHSIZE number_of_pages
Number of 4KB pages that will be read from the table space when data prefetching is being performed.

OVERHEAD number_of_milliseconds
Number that specifies the I/O controller overhead, disk seek, and latency time in milliseconds.

TRANSFERRATE number_of_milliseconds
Number that specifies the time in milliseconds to read one 4 KB page into memory.

comment
Description of the database or the database entry in the system directory. Do not use a carriage return or line feed character in the comment. Be sure to enclose the comment text in double quotation marks. Maximum size is 30 characters.

Related tasks:

- "Creating a database" on page 293

Related reference:

- "CREATE DATABASE " on page 461
- "sqledbdescext data structure" in *Administrative API Reference*
- "sqlecadb - Catalog a database in the system database directory" on page 794
- "sqlecran API - Create a database on a database partition server" in *Administrative API Reference*
- "sqledbterritoryinfo data structure" in *Administrative API Reference*
- "sqledrpd - Drop database" on page 808
- "SQLCA data structure" in *Administrative API Reference*
- "sqledbdesc data structure" in *Administrative API Reference*
- "sqledpan - Drop a database on a database partition server" on page 807

Related samples:

- "db_udcs.cbl -- How to use user-defined collating sequence (IBM COBOL)"
- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "ebcdicdb.cbl -- Create a database with EBCDIC 037 standard collating sequence (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbrecov.sqc -- How to recover a database (C)"
- "dbcreate.C -- Create and drop databases (C++)"

- “dbrecov.sqC -- How to recover a database (C++)”

sqledpan - Drop a database on a database partition server

Drops a database at a specified database partition server. Can only be run in a partitioned database environment.

Scope:

This API only affects the database partition server on which it is called.

Authorization:

One of the following:
- sysadm
- sysctrl

Required connection:

None. An instance attachment is established for the duration of the call.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqledpan (
        char * pDbAlias,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdpan (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        void * pReserved2,
        char * pDbAlias);
```

sqledpan API parameters:

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdpan API-specific parameters:

Reserved1
Reserved for future use.

DbAliasLen
: Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

pReserved2
: A spare pointer that is set to null or points to zero. Reserved for future use.

Usage notes:

Improper use of this API can cause inconsistencies in the system, so it should only be used with caution.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:
- "Calling DB2 APIs in REXX" in *Administrative API Reference*

Related tasks:
- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*
- "Dropping a database" in *Administration Guide: Implementation*

Related reference:
- "DROP DATABASE " on page 494
- "sqlecran API - Create a database on a database partition server" in *Administrative API Reference*
- "sqledrpd - Drop database" on page 808
- "SQLCA data structure" in *Administrative API Reference*

sqledrpd - Drop database

Deletes the database contents and all log files for the database, uncatalogs the database, and deletes the database subdirectory.

Scope:

By default, this API affects all database partition servers that are listed in the db2nodes.cfg file.

Authorization:

One of the following:
- sysadm
- sysctrl

Required connection:

Instance. It is not necessary to call ATTACH before dropping a remote database. If the database is cataloged as remote, an instance attachment to the remote node is established for the duration of the call.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqledrpd (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pReserved2,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpd (
        unsigned short Reserved1,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pReserved2,
        _SQLOLDCHAR * pDbAlias);
```

sqledrpd API parameters:

pDbAlias
Input. A string containing the alias of the database to be dropped. This name is used to reference the actual database name in the system database directory.

pReserved2
A spare pointer that is set to null or points to zero. Reserved for future use.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdrpd API-specific parameters:

Reserved1
Reserved for future use.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes:

The sqledrpd API deletes all user data and log files. If the log files are needed for a roll-forward recovery after a restore operation, the files should be saved prior to calling this API.

The database must not be in use; all users must be disconnected from the database before the database can be dropped.

To be dropped, a database must be cataloged in the system database directory. Only the specified database alias is removed from the system database directory. If other aliases with the same database name exist, their entries remain. If the database being dropped is the last entry in the local database directory, the local database directory is deleted automatically.

If this API is called from a remote client (or from a different instance on the same machine), the specified alias is removed from the client's system database directory. The corresponding database name is removed from the server's system database directory.

This API unlinks all files that are linked through any DATALINK columns. Since the unlink operation is performed asynchronously on the DB2 Data Links Manager, its effects may not be seen immediately on the DB2 Data Links Manager, and the unlinked files may not be immediately available for other operations. When the API is called, all the DB2 Data Links Managers configured to that database must be available; otherwise, the drop database operation will fail.

REXX API syntax:

```
DROP DATABASE dbalias
```

REXX API parameters:

dbalias
The alias of the database to be dropped.

Related tasks:
- "Dropping a database" in *Administration Guide: Implementation*

Related reference:
- "DROP DATABASE " on page 494
- "sqlecrea - Create database" on page 800
- "sqledpan - Drop a database on a database partition server" on page 807
- "sqleuncd API - Uncatalog a database from the system database directory" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

Related samples:
- "dbconf.cbl -- Update database configuration (IBM COBOL)"
- "dbcreate.c -- Create and drop databases (C)"
- "dbcreate.C -- Create and drop databases (C++)"

sqledrpn - Check whether a database partition server can be dropped

Verifies whether a database partition server is being used by a database. A message is returned, indicating whether the database partition server can be dropped.

Scope:

This API only affects the database partition server on which it is issued.

Authorization:

One of the following:
- sysadm
- sysctrl

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqledrpn (
        unsigned short Action,
        void * pReserved,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrpn (
        unsigned short Reserved1,
        struct sqlca * pSqlca,
        void * pReserved2,
        unsigned short Action);
```

sqledrpn API parameters:

Action
The action requested. The valid value is: SQL_DROPNODE_VERIFY

pReserved
Reserved. Should be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgdrpn API-specific parameters:

Reserved1
Reserved for the length of pReserved2.

pReserved2
A spare pointer that is set to NULL or points to 0. Reserved for future use.

Usage notes:

If a message is returned, indicating that the database partition server is not in use, use the db2stop command with DROP NODENUM to remove the entry for the database partition server from the db2nodes.cfg file, which removes the database partition server from the partitioned database environment.

If a message is returned, indicating that the database partition server is in use, the following actions should be taken:

1. The database partition server to be dropped will have a database partition on it for each database in the instance. If any of these database partitions contain data, redistribute the database partition groups that use these database partitions. Redistribute the database partition groups to move the data to database partitions that exist at database partition servers that are not being dropped.
2. After the database partition groups are redistributed, drop the database partition from every database partition group that uses it. To remove a database partition from a database partition group, you can use either the drop node option of the sqludrdt API or the ALTER DATABASE PARTITION GROUP statement.
3. Drop any event monitors that are defined on the database partition server.
4. Rerun sqledrpn to ensure that the database partition at the database partition server is no longer in use.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:

- "Calling DB2 APIs in REXX" in *Administrative API Reference*

Related tasks:

- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*
- "Dropping a database partition" on page 335

Related reference:

- "sqleaddn - Add a database partition server to the partitioned database environment" on page 792
- "SQLCA data structure" in *Administrative API Reference*
- "ALTER DATABASE PARTITION GROUP " on page 845
- "DROP DBPARTITIONNUM VERIFY " on page 495

sqlefrce - Force users and applications off the system

Forces local or remote users or applications off the system to allow for maintenance on a server. Attention: If an operation that cannot be interrupted (a database restore, for example) is forced, the operation must be successfully re-executed before the database becomes available.

Scope:

This API affects all database partition servers that are listed in the db2nodes.cfg file.

In a partitioned database environment, this API does not have to be issued from the coordinator partition of the application being forced. This API can be issued from any database partition server in the partitioned database environment.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint

Required connection:

Instance. To force users off a remote server, it is necessary to first attach to that server. If no attachment exists, this API is executed locally.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlefrce (
        sqlint32 NumAgentIds,
        sqluint32 * pAgentIds,
        unsigned short ForceMode,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgfrce (
        struct sqlca * pSqlca,
        unsigned short ForceMode,
        sqluint32 * pAgentIds,
        sqlint32 NumAgentIds);
```

sqlefrce API parameters:

NumAgentIds
: Input. An integer representing the total number of users to be terminated. This number should be the same as the number of elements in the array of agent IDs.

  If this parameter is set to SQL_ALL_USERS (defined in sqlenv), all applications with either database connections or instance attachments are forced. If it is set to zero, an error is returned.

pAgentIds
: Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding database user.

ForceMode
: Input. An integer specifying the operating mode of the sqlefrce API. Only the asynchronous mode is supported. This means that the API does not wait until all specified users are terminated before returning. It returns as soon as the API has been issued successfully, or an error occurs. As a result, there may be a short interval between the time the force application call completes and the specified users have been terminated.

  This parameter must be set to SQL_ASYNCH (defined in sqlenv).

pSqlca
: Output. A pointer to the sqlca structure.

Usage notes:

The database manager remains active so that subsequent database manager operations can be handled without the need for db2start.

To preserve database integrity, only users who are idling or executing interruptible database operations can be forced off.

After a force command has been issued, the database will still accept requests to connect. Additional forces may be required to completely force all users off. The database system monitor functions are used to gather the agent IDs of the users to be forced.

When the force mode is set to SQL_ASYNCH (the only value permitted), the API immediately returns to the calling application.

Minimal validation is performed on the array of agent IDs to be forced. The user must ensure that the pointer points to an array containing the total number of elements specified. If NumAgentIds is set to SQL_ALL_USERS, the array is ignored.

When a user is forced off, a unit of work rollback is performed to ensure database consistency.

All users that can be forced will be forced. If one or more specified agent IDs cannot be found, sqlcode in the sqlca structure is set to 1230. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and sqlefrce is called. The user that calls this API is never forced off.

Agent IDs are recycled, and are used to force applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the wrong user may be forced.

REXX API syntax:

```
FORCE APPLICATION {ALL | :agentidarray} [MODE ASYNC]
```

REXX API parameters:

ALL All applications will be disconnected. This includes applications that have database connections and applications that have instance attachments.

agentidarray
A compound REXX host variable containing the list of agent IDs to be terminated. In the following, XXX is the name of the host variable:

- XXX.0
Number of agents to be terminated

- XXX.1
First agent ID

- XXX.2
Second agent ID

- XXX.3
and so on.

ASYNC
The only mode currently supported means that sqlefrce does not wait until all specified applications are terminated before returning.

Related reference:

- "FORCE APPLICATION command" in *Command Reference*
- "FORCE APPLICATION command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "sqleatin - Attach to instance" on page 838
- "sqledtin - Detach from instance" on page 841
- "SQLCA data structure" in *Administrative API Reference*

Related samples:

- "dbstop.cbl -- How to stop a database manager (IBM COBOL)"
- "dbsample.sqc -- Creates a sample database (C)"
- "dbconn.sqc -- How to connect to and disconnect from a database (C)"
- "instart.c -- Stop and start the current local instance (C)"
- "dbconn.sqC -- How to connect to and disconnect from a database (C++)"
- "instart.C -- Stop and start the current local instance (C++)"

sqlemgdb - Migrate previous version of DB2 database to current version

Converts previous (Version 8.x or higher) versions of DB2 databases to current versions.

Authorization:

sysadm

Required connection:

This API establishes a database connection.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlemgdb (
        _SQLOLDCHAR * pDbAlias,
        _SQLOLDCHAR * pUserName,
        _SQLOLDCHAR * pPassword,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgmgdb (
        unsigned short PasswordLen,
        unsigned short UserNameLen,
        unsigned short DbAliasLen,
        struct sqlca * pSqlca,
        _SQLOLDCHAR * pPassword,
        _SQLOLDCHAR * pUserName,
        _SQLOLDCHAR * pDbAlias);
```

sqlemgdb API parameters:

pDbAlias
Input. A string containing the alias of the database that is cataloged in the system database directory.

pUserName
Input. A string containing the user name of the application. May be NULL.

pPassword
Input. A string containing the password of the supplied user name (if any). May be NULL.

pSqlca
Output. A pointer to the sqlca structure.

sqlgmgdb API-specific parameters:

PasswordLen
Input. A 2-byte unsigned integer representing the length in bytes of the password. Set to zero when no password is supplied.

UserNameLen
Input. A 2-byte unsigned integer representing the length in bytes of the user name. Set to zero when no user name is supplied.

DbAliasLen
Input. A 2-byte unsigned integer representing the length in bytes of the database alias.

Usage notes:

This API will only migrate a database to a newer version, and cannot be used to convert a migrated database to its previous version.

The database must be cataloged before migration.

REXX API syntax:

```
MIGRATE DATABASE dbalias [USER username USING password]
```

REXX API parameters:

dbalias
Alias of the database to be migrated.

username
User name under which the database is to be restarted.

password
Password used to authenticate the user name.

Related reference:
- "MIGRATE DATABASE " on page 566
- "SQLCA data structure" in *Administrative API Reference*

Related samples:
- "migrate.cbl -- Demonstrates how to migrate to a database (IBM COBOL)"
- "dbmigrat.c -- Migrate a database (C)"
- "dbmigrat.C -- Migrate a database (C++)"

sqlesdeg - Set the maximum runtime intra-partition parallelism level or degree for SQL statements

Sets the maximum run time degree of intra-partition parallelism for SQL statement execution for specified active applications. It has no effect on CREATE INDEX statement execution parallelism.

Scope:

This API affects all database partition servers that are listed in the db2nodes.cfg file.

Authorization:

One of the following:
- sysadm
- sysctrl

Required connection:

Instance. To change the maximum run time degree of parallelism on a remote server, it is first necessary to attach to that server. If no attachment exists, the SET RUNTIME DEGREE statement fails.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlesdeg (
        sqlint32 NumAgentIds,
        sqluint32 * pAgentIds,
        sqlint32 Degree,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgsdeg (
        struct sqlca * pSqlca,
        sqlint32 Degree,
        sqluint32 * pAgentIds,
        sqlint32 NumAgentIds);
```

sqlesdeg API parameters:

NumAgentIds
: Input. An integer representing the total number of active applications to which the new degree value will apply. This number should be the same as the number of elements in the array of agent IDs.

  If this parameter is set to SQL_ALL_USERS (defined in sqlenv), the new degree will apply to all active applications. If it is set to zero, an error is returned.

pAgentIds
: Input. Pointer to an array of unsigned long integers. Each entry describes the agent ID of the corresponding application. To list the agent IDs of the active applications, use the db2GetSnapshot API.

Degree
: Input. The new value for the maximum run time degree of parallelism. The value must be in the range 1 to 32767.

pSqlca
: Output. A pointer to the sqlca structure.

Usage notes:

The database system monitor functions are used to gather the agent IDs and degrees of active applications.

Minimal validation is performed on the array of agent IDs. The user must ensure that the pointer points to an array containing the total number of elements specified. If NumAgentIds is set to SQL_ALL_USERS, the array is ignored.

If one or more specified agent IDs cannot be found, the unknown agent IDs are ignored, and the function continues. No error is returned. An agent ID may not be found, for instance, if the user signs off between the time an agent ID is collected and the API is called.

Agent IDs are recycled, and are used to change the degree of parallelism for applications some time after being gathered by the database system monitor. When a user signs off, therefore, another user may sign on and acquire the same agent ID through this recycling process, with the result that the new degree of parallelism will be modified for the wrong user.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related tasks:

- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:

- "SET RUNTIME DEGREE " on page 615
- "db2GetSnapshot API - Get a snapshot of the database manager operational status" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

Related samples:

- "ininfo.c -- Set and get information at the instance level (C)"
- "ininfo.C -- Set and get information at the instance level (C++)"

sqluadau - Get current user's authorities

Reports the instance level and database level authorities of the current user from values found in the database manager configuration file and the authorization system catalog view (SYSCAT.DBAUTH) respectively. The instance level authorities reported are the ones that you can set in the sysadm_group, sysmaint_group and sysctrl_group database manager configuration parameters and the database level authorities are the ones that can be granted via the GRANT (Database Authorities) statement.

Authorization:

None

Required connection:

Database

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqluadau (
   struct sql_authorizations * pAuthorizations,
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgadau (
   struct sql_authorizations * pAuthorizations,
   struct sqlca * pSqlca);
```

sqluadau API parameters:

pAuthorizations
Input or Output. Pointer to the sql_authorizations structure. This array of short integers indicates which authorizations the current user holds.

The first element in the structure, sql_authorizations_len, must be initialized to the size of the buffer being passed, prior to calling this API.

pSqlca
Output. A pointer to the sqlca structure.

Usage notes:

Direct authorities are acquired by explicit commands that grant the authorities to a user ID. Indirect authorities are based on authorities acquired by the groups to which a user belongs.

Note: PUBLIC is a special group to which all users belong.

If there are no errors, each element of the sql_authorizations structure contains a 0 or a 1. A value of 1 indicates that the user holds that authorization; 0 indicates that the user does not.

REXX API syntax:

```
GET AUTHORIZATIONS :value
```

REXX API parameters:

value A compound REXX host variable to which the authorization level is returned. In the following, XXX represents the host variable name. Values are 0 for no, and 1 for yes.

XXX.0	Number of elements in the variable (always 18)
XXX.1	Direct SYSADM authority
XXX.2	Direct DBADM authority
XXX.3	Direct CREATETAB authority
XXX.4	Direct BINDADD authority
XXX.5	Direct CONNECT authority
XXX.6	Indirect SYSADM authority
XXX.7	Indirect DBADM authority
XXX.8	Indirect CREATETAB authority
XXX.9	Indirect BINDADD authority
XXX.10	Indirect CONNECT authority
XXX.11	Direct SYSCTRL authority
XXX.12	Indirect SYSCTRL authority

XXX.13
Direct SYSMAINT authority

XXX.14
Indirect SYSMAINT authority

XXX.15
Direct CREATE_NOT_FENC authority

XXX.16
Indirect CREATE_NOT_FENC authority

XXX.17
Direct IMPLICIT_SCHEMA authority

XXX.18
Indirect IMPLICIT_SCHEMA authority.

XXX.19
Direct LOAD authority.

XXX.20
Indirect LOAD authority.

Related reference:
- "sql_authorizations " on page 1370
- "SQLCA data structure" in *Administrative API Reference*

sqludrdt - Redistribute data across a database partition group

Redistributes data across the database partitions in a database partition group. The current data distribution, whether it is uniform or skewed, can be specified. The redistribution algorithm selects the data basepartitions to be moved based on the current data distribution.

This API can only be called from the catalog partition. Use the LIST DATABASE DIRECTORY command to determine which database partition server is the catalog partition for each database.

Scope:

This API affects all database partitions in the database partition group.

Authorization:

One of the following:
- sysadm
- sysctrl (DELETE, INSERT, and UPDATE authority is also required on all tables in the database partition group being redistributed)
- dbadm

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqludrdt (
   char * pNodeGroupName,
   char * pTargetPMapFileName,
   char * pDataDistFileName,
   SQL_PDB_NODE_TYPE * pAddList,
   unsigned short AddCount,
   SQL_PDB_NODE_TYPE * pDropList,
   unsigned short DropCount,
   unsigned char DataRedistOption,
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdrdt (
   unsigned short NodeGroupNameLen,
   unsigned short TargetPMapFileNameLen,
   unsigned short DataDistFileNameLen,
   char * pNodeGroupName,
   char * pTargetPMapFileName,
   char * pDataDistFileName,
   SQL_PDB_NODE_TYPE * pAddList,
   unsigned short AddCount,
   SQL_PDB_NODE_TYPE * pDropList,
   unsigned short DropCount,
   unsigned char DataRedistOption,
   struct sqlca * pSqlca);
```

sqludrdt API parameters:

pNodeGroupName
The name of the database partition group to be redistributed.

pTargetPMapFileName
The name of the file that contains the target distribution map. If a directory path is not specified as part of the file name, the current directory is used. This parameter is used when the DataRedistOption value is T. The file should be in character format and contain either 4 096 entries (for a multiple-partition database partition group) or 1 entry (for a single-partition database partition group). Entries in the file indicate node numbers. Entries can be in free format.

pDataDistFileName
The name of the file that contains input distribution information. If a directory path is not specified as part of the file name, the current directory is used. This parameter is used when the DataRedistOption value is U. The file should be in character format and contain 4 096 positive integer entries. Each entry in the file should indicate the weight of the corresponding database partition. The sum of the 4 096 values should be less than or equal to 4 294 967 295.

pAddList
The list of database partitions to add to the database partition group during the data redistribution. Entries in the list must be in the form: SQL_PDB_NODE_TYPE.

AddCount
The number of database partitions to add to the database partition group.

pDropList
The list of database partitions to drop from the database partition group during the data redistribution. Entries in the list must be in the form: SQL_PDB_NODE_TYPE.

DropCount
The number of database partitions to drop from the database partition group.

DataRedistOption

A single character that indicates the type of data redistribution to be done. Possible values are:

U Specifies to redistribute the database partition group to achieve a balanced distribution. If pDataDistFileName is null, the current data distribution is assumed to be uniform (that is, each database partition represents the same amount of data). If pDataDistFileName parameter is not null, the values in this file are assumed to represent the current data distribution. When the DataRedistOption is U, the pTargetPMapFileName parameter should be null. Database partitions specified in the add list are added, and database partitions specified in the drop list are dropped from the database partition group.

T Specifies to redistribute the database partition group using the pTargetPMapFileName parameter. For this option, the parameters, pDataDistFileName, pAddList, and pDropList should be null, and both the parameters, AddCount and DropCount must be zero.

C Specifies to continue a redistribution operation that failed. For this option, the parameters, pTargetPMapFileName, pDataDistFileName, pAddList, and pDropList should be null, and both the parameters, AddCount and DropCount must be zero.

R Specifies to roll back a redistribution operation that failed. For this option, the parameters, pTargetPMapFileName, pDataDistFileName, pAddList, and pDropList should be null, and both the parameters, AddCount and DropCount must be zero.

pSqlca

Output. A pointer to the sqlca structure.

sqlgdrdt API-specific parameters:

NodeGroupNameLen

The length of the name of the database partition group.

TargetPMapFileNameLen

The length of the name of the target distribution map file.

DataDistFileNameLen

The length of the name of the data distribution file.

Usage notes:

When a redistribution operation is done, a message file is written to:

- The $HOME/sqllib/redist directory on UNIX based systems, using the following format for subdirectories and file name: database-name.nodegroup-name.timestamp.
- The $HOME\sqllib\redist\ directory on the Windows operating system, using the following format for subdirectories and file name: database-name\first-eight-characters-of-the-nodegroup-name\date\time.

The time stamp value is the time at which the API was called.

This utility performs intermittent COMMITs during processing.

Use the ALTER DATABASE PARTITION GROUP statement to add database partitions to a database partition group. This statement permits one to define the containers for the table spaces associated with the database partition group.

All packages having a dependency on a table that has undergone redistribution are invalidated. It is recommended to explicitly rebind such packages after the redistribute database partition group operation has completed. Explicit rebinding eliminates the initial delay in the execution of the first SQL request for the invalid package. The redistribute message file contains a list of all the tables that have undergone redistribution.

It is also recommended to update statistics by issuing the db2Runstats API after the redistribute database partition group operation has completed.

Database partition groups containing replicated summary tables or tables defined with the DATA CAPTURE CHANGES clause cannot be redistributed.

Redistribution is not allowed if there are user temporary table spaces with existing declared temporary tables in the database partition group.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:
- "Calling DB2 APIs in REXX" in *Administrative API Reference*

Related tasks:
- "Considerations while programming REXX embedded SQL applications" in *Developing Embedded SQL Applications*
- "Redistributing data across database partitions" on page 339

Related reference:
- "db2Runstats API - Update statistics for tables and indexes" in *Administrative API Reference*
- "sqlarbnd - Rebind package" on page 833
- "SQLCA data structure" in *Administrative API Reference*
- "REDISTRIBUTE DATABASE PARTITION GROUP command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "ALTER DATABASE PARTITION GROUP " on page 845
- "LIST DATABASE DIRECTORY command" in *Command Reference*
- "REDISTRIBUTE DATABASE PARTITION GROUP " on page 577

sqlugrpn - Get the database partition server number for a row

Returns the database partition number and the database partition server number based on the distribution key values. An application can use this information to determine on which database partition server a specific row of a table is stored.

The partitioning data structure, sqlupi, is the input for this API. The structure can be returned by the sqlugtpi API. Another input is the character representations of the corresponding distribution key values. The output is a database partition number generated by the distribution strategy and the corresponding database partition server number from the distribution map. If the distribution map information is not provided, only the database partition number is returned. This can be useful when analyzing data distribution.

The database manager does not need to be running when this API is called.

Scope:

This API must be invoked from a database partition server in the db2nodes.cfg file. This API should not be invoked from a client, since it could result in erroneous database partitioning information being returned due to differences in codepage and endianess between the client and the server.

Authorization:

None

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlugrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_ctrycode,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);

SQL_API_RC SQL_API_FN
  sqlggrpn (
   unsigned short num_ptrs,
   unsigned char ** ptr_array,
   unsigned short * ptr_lens,
   unsigned short territory_code,
   unsigned short codepage,
   struct sqlupi * part_info,
   short * part_num,
   SQL_PDB_NODE_TYPE * node_num,
   unsigned short chklvl,
   struct sqlca * sqlca,
   short dataformat,
   void * pReserved1,
   void * pReserved2);
```

sqlugrpn API parameters:

num_ptrs
The number of pointers in ptr_array. The value must be the same as the one specified for the part_info parameter; that is, part_info->sqld.

ptr_array
An array of pointers that points to the character representations of the corresponding values of each part of the distribution key specified in

part_info. If a null value is required, the corresponding pointer is set to null. For generated columns, this function does not generate values for the row. The user is responsible for providing a value that will lead to the correct partitioning of the row.

ptr_lens
An array of unsigned integers that contains the lengths of the character representations of the corresponding values of each part of the partitioning key specified in part_info.

territory_ctrycode
The country/region code of the target database. This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

codepage
The code page of the target database. This value can also be obtained from the database configuration file using the GET DATABASE CONFIGURATION command.

part_info
A pointer to the sqlupi structure.

part_num
A pointer to a 2-byte signed integer that is used to store the database partition number.

node_num
A pointer to an SQL_PDB_NODE_TYPE field used to store the node number. If the pointer is null, no node number is returned.

chklvl An unsigned integer that specifies the level of checking that is done on input parameters. If the value specified is zero, no checking is done. If any non-zero value is specified, all input parameters are checked.

sqlca Output. A pointer to the sqlca structure.

dataformat
Specifies the representation of distribution key values. Valid values are:

SQL_CHARSTRING_FORMAT
All distribution key values are represented by character strings. This is the default value.

SQL_IMPLIEDDECIMAL_FORMAT
The location of an implied decimal point is determined by the column definition. For example, if the column definition is DECIMAL(8,2), the value 12345 is processed as 123.45.

SQL_PACKEDDECIMAL_FORMAT
All decimal column distribution key values are in packed decimal format.

SQL_BINARYNUMERICS_FORMAT
All numeric distribution key values are in big-endian binary format.

pReserved1
Reserved for future use.

pReserved2
Reserved for future use.

Usage notes:

Data types supported on the operating system are the same as those that can be defined as a distribution key.

Note: CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types must be converted to the database code page before this API is called.

For numeric and datetime data types, the character representations must be at the code page of the respective system where the API is invoked.

If node_num is not null, the distribution map must be supplied; that is, pmaplen field in part_info parameter (part_info->pmaplen) is either 2 or 8192. Otherwise, SQLCODE -6038 is returned. The distribution key must be defined; that is, sqld field in part_info parameter (part_info->sqld) must be greater than zero. Otherwise, SQLCODE -2032 is returned.

If a null value is assigned to a non-nullable partitioning column, SQLCODE -6039 is returned.

All the leading blanks and trailing blanks of the input character string are stripped, except for the CHAR, VARCHAR, GRAPHIC, and VARGRAPHIC data types, where only trailing blanks are stripped.

Related reference:
- "GET DATABASE CONFIGURATION " on page 502
- "sqlugtpi - Get table distribution information" on page 826
- "sqludrdt - Redistribute data across a database partition group" on page 820
- "SQLCA data structure" in *Administrative API Reference*
- "sqlupi data structure" in *Administrative API Reference*
- "db2CfgGet - Get the database manager or database configuration parameters" on page 672

sqlugtpi - Get table distribution information

Allows an application to obtain the distribution information for a table. The distribution information includes the distribution map and the column definitions of the distribution key. Information returned by this API can be passed to the sqlugrpn API to determine the database partition number and the database partition server number for any row in the table.

To use this API, the application must be connected to the database that contains the table for which distribution information is being requested.

Scope:

This API can be executed on any database partition server defined in the db2nodes.cfg file.

Authorization:

For the table being referenced, a user must have at least one of the following:
- sysadm authority

- dbadm authority
- CONTROL privilege
- SELECT privilege

Required connection:

Database

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlugtpi (
   unsigned char * tablename,
   struct sqlupi * part_info,
   struct sqlca * sqlca);

SQL_API_RC SQL_API_FN
  sqlggtpi (
   unsigned short tn_length,
   unsigned char * tablename,
   struct sqlupi * part_info,
   struct sqlca * sqlca);
```

sqlugtpi API parameters:

tablename
: The fully qualified name of the table.

part_info
: A pointer to the sqlupi structure.

pSqlca
: Output. A pointer to the sqlca structure.

sqlggtpi API-specific parameters:

tn_length
: A 2-byte unsigned integer with the length of the table name.

Related reference:

- "sqludrdt - Redistribute data across a database partition group" on page 820
- "sqlugrpn - Get the database partition server number for a row" on page 823
- "SQLCA data structure" in *Administrative API Reference*
- "sqlupi data structure" in *Administrative API Reference*

sqluvqdp - Quiesce table spaces for a table

Quiesces table spaces for a table. There are three valid quiesce modes: share, intent to update, and exclusive. There are three possible table space states resulting from the quiesce function:

- Quiesced: SHARE
- Quiesced: UPDATE
- Quiesced: EXCLUSIVE

Scope:

sqluvqdp - Quiesce table spaces for a table

In a single-partition database environment, this API quiesces all table spaces involved in a load operation in exclusive mode for the duration of the load. In a partitioned database environment, this API acts locally on a database partition. It quiesces only that portion of table spaces belonging to the database partition on which the load is performed.

Authorization:

One of the following:

- sysadm
- sysctrl
- sysmaint
- dbadm
- load

Required connection:

Database

API include file:

```
sqlutil.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqluvqdp (
   char * pTableName,
   sqlint32 QuiesceMode,
   void * pReserved,
   struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgvqdp (
   unsigned short TableNameLen,
   char * pTableName,
   sqlint32 QuiesceMode,
   void * pReserved,
   struct sqlca * pSqlca);
```

sqluvqdp API parameters:

pTableName
: Input. A string containing the table name as used in the system catalog. This may be a two-part name with the schema and the table name separated by a period (.). If the schema is not provided, the CURRENT SCHEMA will be used.

  The table cannot be a system catalog table. This field is mandatory.

QuiesceMode
: Input. Specifies the quiesce mode. Valid values (defined in sqlutil) are:

  SQLU_QUIESCEMODE_SHARE
  : For share mode

  SQLU_QUIESCEMODE_INTENT_UPDATE
  : For intent to update mode

  SQLU_QUIESCEMODE_EXCLUSIVE
  : For exclusive mode

SQLU_QUIESCEMODE_RESET
: To reset the state of the table spaces to normal if either of the following is true:
  - The caller owns the quiesce
  - The caller who sets the quiesce disconnects, creating a ″phantom quiesce″

SQLU_QUIESCEMODE_RESET_OWNED
: To reset the state of the table spaces to normal if the caller owns the quiesce.

This field is mandatory.

pReserved
: Reserved for future use.

pSqlca
: Output. A pointer to the sqlca structure.

sqlgvqdp API-specific parameters:

TableNameLen
: Input. A 2-byte unsigned integer representing the length in bytes of the table name.

Usage notes:

This API is not supported for declared temporary tables.

When the quiesce share request is received, the transaction requests intent share locks for the table spaces and a share lock for the table. When the transaction obtains the locks, the state of the table spaces is changed to QUIESCED SHARE. The state is granted to the quiescer only if there is no conflicting state held by other users. The state of the table spaces is recorded in the table space table, along with the authorization ID and the database agent ID of the quiescer, so that the state is persistent.

The table cannot be changed while the table spaces for the table are in QUIESCED SHARE state. Other share mode requests to the table and table spaces will be allowed. When the transaction commits or rolls back, the locks are released, but the table spaces for the table remain in QUIESCED SHARE state until the state is explicitly reset.

When the quiesce exclusive request is made, the transaction requests super exclusive locks on the table spaces, and a super exclusive lock on the table. When the transaction obtains the locks, the state of the table spaces changes to QUIESCED EXCLUSIVE. The state of the table spaces, along with the authorization ID and the database agent ID of the quiescer, are recorded in the table space table. Since the table spaces are held in super exclusive mode, no other access to the table spaces is allowed. The user who invokes the quiesce function (the quiescer), however, has exclusive access to the table and the table spaces.

When a quiesce update request is made, the table spaces are locked in intent exclusive (IX) mode, and the table is locked in update (U) mode. The state of the table spaces with the quiescer is recorded in the table space table.

There is a limit of five quiescers on a table space at any given time. Since QUIESCED EXCLUSIVE is incompatible with any other state, and QUIESCED UPDATE is incompatible with another QUIESCED UPDATE, the five quiescer limit, if reached, must have at least four QUIESCED SHARE and at most one QUIESCED UPDATE.

A quiescer can upgrade the state of a table space from a less restrictive state to a more restrictive one (for example, S to U, or U to X). If a user requests a state lower than one that is already held, the original state is returned. States are not downgraded.

The quiesced state of a table space must be reset explicitly by using SQLU_QUIESCEMODE_RESET.

REXX API syntax:

```
QUIESCE TABLESPACES FOR TABLE table_name
{SHARE | INTENT TO UPDATE | EXCLUSIVE | RESET}
```

REXX API parameters:

table_name
Name of the table as used in the system catalog. This may be a two-part name with the schema and the table name separated by a period (.). If the schema is not provided, the CURRENT SCHEMA will be used.

Related reference:
- "QUIESCE TABLESPACES FOR TABLE " on page 569
- "db2DatabaseQuiesce - Quiesce the database" on page 681
- "db2InstanceQuiesce - Quiesce instance" on page 712
- "SQLCA data structure" in *Administrative API Reference*
- "QUIESCE TABLESPACES FOR TABLE command using the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*

Related samples:
- "tload.sqb -- How to export and load table data (IBM COBOL)"
- "tbmove.sqc -- How to move table data (C)"
- "tbmove.sqC -- How to move table data (C++)"

Chapter 39. DB2 UDB APIs for Users

Following are the application programming interfaces (APIs) that correspond to the DB2 UDB commands that are used for the Common Criteria evaluation. Note that:

- APIs are not used for the Common Criteria certification. The APIs are included in this document for reasons of completeness only.
- Not every API has a corresponding command, and vice versa.

sqlaprep - Precompile application program

Processes an application program source file containing embedded SQL statements. A modified source file is produced containing host language calls for the SQL statements and, by default, a package is created in the database.

Scope:

This API can be called from any database partition server in db2nodes.cfg. It updates the database catalogs on the catalog partition. Its effects are visible to all database partition servers.

Authorization:

One of the following:

- sysadm or dbadm authority
- BINDADD privilege if a package does not exist and one of:
- IMPLICIT_SCHEMA authority on the database if the schema name of the package does not exist
- CREATEIN privilege on the schema if the schema name of the package exists
- ALTERIN privilege on the schema if the package exists
- BIND privilege on the package if it exists.

The user also needs all privileges required to compile any static SQL statements in the application. Privileges granted to groups are not used for authorization checking of static statements. If the user has sysadm authority, but not explicit privileges to complete the bind, the database manager grants explicit dbadm authority automatically.

Required connection:

Database

API include file:

`sql.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlaprep (
        _SQLOLDCHAR * pProgramName,
        _SQLOLDCHAR * pMsgFileName,
        struct sqlopt * pPrepOptions,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgprep (
        unsigned short MsgFileNameLen,
        unsigned short ProgramNameLen,
        struct sqlca * pSqlca,
        struct sqlopt * pPrepOptions,
        _SQLOLDCHAR * pMsgFileName,
        _SQLOLDCHAR * pProgramName);
```

sqlaprep API parameters:

pProgramName
Input. A string containing the name of the application to be precompiled. Use the following extensions:
- .sqb: for COBOL applications
- .sqc: for C applications
- .sqC: for UNIX C++ applications
- .sqf: for FORTRAN applications
- .sqx: for C++ applications

When the TARGET option is used, the input file name extension does not have to be from this predefined list.

The preferred extension for C++ applications containing embedded SQL on UNIX based systems is sqC; however, the sqx convention, which was invented for systems that are not case sensitive, is tolerated by UNIX based systems.

pMsgFileName
Input. A string containing the destination for error, warning, and informational messages. Can be the path and the name of an operating system file, or a standard device. If a file already exists, it is overwritten. If it does not exist, a file is created.

pPrepOptions
Input. A structure used to pass precompile options to the API. For more information about this structure, see SQLOPT.

pSqlca
Output. A pointer to the sqlca structure.

sqlgprep API-specific parameters:

MsgFileNameLen
Input. Length in bytes of the pMsgFileName parameter.

ProgramNameLen
Input. Length in bytes of the pProgramName parameter.

Usage notes:

A modified source file is produced, which contains host language equivalents to the SQL statements. By default, a package is created in the database to which a connection has been established. The name of the package is the same as the program file name (minus the extension and folded to uppercase), up to a maximum of 8 characters.

Following connection to a database, sqlaprep executes under the transaction that was started. PRECOMPILE PROGRAM then issues a COMMIT or a ROLLBACK operation to terminate the current transaction and start another one.

Precompiling stops if a fatal error or more than 100 errors occur. If a fatal error does occur, PRECOMPILE PROGRAM stops precompiling, attempts to close all files, and discards the package.

The Precompile option types and values are defined in sql.h.

REXX API syntax:

```
This API can be called from REXX through the SQLDB2 interface.
```

Related concepts:

- "Calling DB2 APIs in REXX" in *Administrative API Reference*

Related reference:

- "sqlabndx - Bind application program to create a package" on page 779
- "SQLCA data structure" in *Administrative API Reference*
- "sqlopt data structure" in *Administrative API Reference*
- "PRECOMPILE " on page 635

Related samples:

- "dbpkg.sqc -- How to work with packages (C)"
- "dbpkg.sqC -- How to work with packages (C++)"

sqlarbnd - Rebind package

Allows the user to recreate a package stored in the database without the need for a bind file.

Authorization:

One of the following:

- sysadm or dbadm authority
- ALTERIN privilege on the schema
- BIND privilege on the package.

The authorization ID logged in the BOUNDBY column of the SYSCAT.PACKAGES system catalog table, which is the ID of the most recent binder of the package, is used as the binder authorization ID for the rebind, and for the default schema for table references in the package. Note that this default qualifier may be different from the authorization ID of the user executing the rebind request. REBIND will use the same bind options that were specified when the package was created.

Required connection:

Database

API include file:

```
sql.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqlarbnd (
        char * pPackageName,
        struct sqlca * pSqlca,
        struct sqlopt * pRebindOptions);

SQL_API_RC SQL_API_FN
  sqlgrbnd (
        unsigned short PackageNameLen,
        char * pPackageName,
        struct sqlca * pSqlca,
        struct sqlopt * pRebindOptions);
```

sqlarbnd API parameters:

pPackageName
: Input. A string containing the qualified or unqualified name that designates the package to be rebound. An unqualified package-name is implicitly qualified by the current authorization ID. This name does not include the package version. When specifying a package that has a version that is not the empty string, then the version-id must be specified using the SQL_VERSION_OPT rebind option.

pSqlca
: Output. A pointer to the sqlca structure.

pRebindOptions
: Input. A pointer to the SQLOPT structure, used to pass rebind options to the API. For more information about this structure, see SQLOPT.

sqlgrbnd API-specific parameters:

PackageNameLen
: Input. Length in bytes of the pPackageName parameter.

Usage notes:

REBIND does not automatically commit the transaction following a successful rebind. The user must explicitly commit the transaction. This enables ″what if ″ analysis, in which the user updates certain statistics, and then tries to rebind the package to see what changes. It also permits multiple rebinds within a unit of work.

This API:

- Provides a quick way to recreate a package. This enables the user to take advantage of a change in the system without a need for the original bind file.fs. For example, if it is likely that a particular SQL statement can take advantage of a newly created index, REBIND can be used to recreate the package. REBIND can also be used to recreate packages after db2Runstats has been executed, thereby taking advantage of the new statistics.
- Provides a method to recreate inoperative packages. Inoperative packages must be explicitly rebound by invoking either the bind utility or the rebind utility. A package will be marked inoperative (the VALID column of the

SYSCAT.PACKAGES system catalog will be set to X) if a function instance on which the package depends is dropped. The rebind conservative option is not supported for inoperative packages.

- Gives users control over the rebinding of invalid packages. Invalid packages will be automatically (or implicitly) rebound by the database manager when they are executed. This may result in a noticeable delay in the execution of the first SQL request for the invalid package. It may be desirable to explicitly rebind invalid packages, rather than allow the system to automatically rebind them, in order to eliminate the initial delay and to prevent unexpected SQL error messages which may be returned in case the implicit rebind fails. For example, following migration, all packages stored in the database will be invalidated by the DB2 Version 5 migration process. Given that this may involve a large number of packages, it may be desirable to explicitly rebind all of the invalid packages at one time. This explicit rebinding can be accomplished using BIND, REBIND, or the db2rbind tool.

The choice of whether to use BIND or REBIND to explicitly rebind a package depends on the circumstances. It is recommended that REBIND be used whenever the situation does not specifically require the use of BIND, since the performance of REBIND is significantly better than that of BIND. BIND must be used, however:

- When there have been modifications to the program (for example, when SQL statements have been added or deleted, or when the package does not match the executable for the program).
- When the user wishes to modify any of the bind options as part of the rebind. REBIND does not support any bind options. For example, if the user wishes to have privileges on the package granted as part of the bind process, BIND must be used, since it has an SQL_GRANT_OPT option.
- When the package does not currently exist in the database.
- When detection of all bind errors is desired. REBIND only returns the first error it detects, and then ends, whereas the BIND command returns the first 100 errors that occur during binding.

REBIND is supported by DB2 Connect.

If REBIND is executed on a package that is in use by another user, the rebind will not occur until the other user's logical unit of work ends, because an exclusive lock is held on the package's record in the SYSCAT.PACKAGES system catalog table during the rebind.

When REBIND is executed, the database manager recreates the package from the SQL statements stored in the SYSCAT.STATEMENTS system catalog table. If many versions with the same package number and creator exist, only one version can be bound at once. If not specified using the SQL_VERSION_OPT rebind option, the VERSION defaults to be "". Even if there is only one package with a name and creator that matches the name and creator specified in the rebind request, it will not rebound unless its VERSION matches the VERSION specified explicitly or implicitly.

If REBIND encounters an error, processing stops, and an error message is returned.

The Explain tables are populated during REBIND if either SQL_EXPLSNAP_OPT or SQL_EXPLAIN_OPT have been set to YES or ALL (check EXPLAIN_SNAPSHOT and EXPLAIN_MODE columns in the catalog). The Explain tables used are those of the REBIND requester, not the original binder. The Rebind option types and values are defined in sql.h.

REXX API syntax:

This API can be called from REXX through the SQLDB2 interface.

Related concepts:
- “Calling DB2 APIs in REXX” in *Administrative API Reference*

Related tasks:
- “Considerations while programming REXX embedded SQL applications” in *Developing Embedded SQL Applications*

Related reference:
- “db2rbind - Rebind all packages ” on page 489
- “REBIND ” on page 659
- “REBIND_ROUTINE_PACKAGE procedure” in *Administrative SQL Routines and Views*
- “db2Runstats API - Update statistics for tables and indexes” in *Administrative API Reference*
- “sqlabndx - Bind application program to create a package” on page 779
- “sqlaprep - Precompile application program” on page 831
- “sqlopt data structure” in *Administrative API Reference*
- “SQLCA data structure” in *Administrative API Reference*

Related samples:
- “dbpkg.sqc -- How to work with packages (C)”
- “dbpkg.sqC -- How to work with packages (C++)”
- “rebind.sqb -- How to rebind a package (IBM COBOL)”

sqleatcp - Attach to instance and change password

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the DB2INSTANCE environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Note: This API extends the function of the sqleatin API by permitting the optional change of the user password for the instance being attached.

Authorization:

None

Required connection:

This API establishes an instance attachment.

API include file:

`sqlenv.h`

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqleatcp (
        char * pNodeName,
        char * pUserName,
        char * pPassword,
        char * pNewPassword,
        struct sqlca * pSqlca);
```

sqleatcp API parameters:

pNodeName
: Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the DB2INSTANCE environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory. May be NULL.

pUserName
: Input. A string containing the user name under which the attachment is to be authenticated. May be NULL.

pPassword
: Input. A string containing the password for the specified user name. May be NULL.

pNewPassword
: Input. A string containing the new password for the specified user name. Set to NULL if a password change is not required.

pSqlca
: Output. A pointer to the sqlca structure.

Usage notes:

A node name in the node directory can be regarded as an alias for an instance.

If an attach request succeeds, the sqlerrmc field of the sqlca will contain 9 tokens separated by hexadecimal FF (similar to the tokens returned when a CONNECT request is successful):

1. Country/region code of the application server
2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the API)
5. Identity and platform type of the server
6. Agent ID of the agent which has been started at the server
7. Agent index
8. Node number of the server
9. Number of database partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the sqlerrmc field of the sqlca (as outlined above).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the DB2INSTANCE environment variable.

Certain functions (db2start, db2stop, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the DB2INSTANCE environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated, and where the password is changed, depend on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

REXX API syntax:

Calling this API directly from REXX is not supported. However, REXX programmers can utilize this function by calling the DB2 command line processor to execute the ATTACH command.

Related reference:
- "ATTACH " on page 631
- "sqleatin - Attach to instance" on page 838
- "sqledtin - Detach from instance" on page 841
- "sqlesetc API - Set client connection settings" in *Administrative API Reference*
- "SQLCA data structure" in *Administrative API Reference*

Related samples:
- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"

sqleatin - Attach to instance

Enables an application to specify the node at which instance-level functions (CREATE DATABASE and FORCE APPLICATION, for example) are to be executed. This node may be the current instance (as defined by the value of the DB2INSTANCE environment variable), another instance on the same workstation, or an instance on a remote workstation. Establishes a logical instance attachment to the node specified, and starts a physical communications connection to the node if one does not already exist.

Note: If a password change is required, use the sqleatcp API instead of the sqleatin API.

Authorization:

None

Required connection:

This API establishes an instance attachment.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqleatin (
        char * pNodeName,
        char * pUserName,
        char * pPassword,
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgatin (
        unsigned short PasswordLen,
        unsigned short UserNameLen,
        unsigned short NodeNameLen,
        struct sqlca * pSqlca,
        char * pPassword,
        char * pUserName,
        char * pNodeName);
```

sqleatin API parameters:

pNodeName
: Input. A string containing the alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the DB2INSTANCE environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory. Can be NULL.

pUserName
: Input. A string containing the user name under which the attachment is to be authenticated. Can be NULL.

pPassword
: Input. A string containing the password for the specified user name. Can be NULL.

pSqlca
: Output. A pointer to the sqlca structure.

sqlgatin API-specific parameters:

PasswordLen
: Input. A 2-byte unsigned integer representing the length of the password in bytes. Set to zero if no password is supplied.

UserNameLen
: Input. A 2-byte unsigned integer representing the length of the user name in bytes. Set to zero if no user name is supplied.

NodeNameLen
: Input. A 2-byte unsigned integer representing the length of the node name in bytes. Set to zero if no node name is supplied.

Usage notes:

Note: A node name in the node directory can be regarded as an alias for an instance.

If an attach request succeeds, the sqlerrmc field of the sqlca will contain 9 tokens separated by hexadecimal FF (similar to the tokens returned when a CONNECT request is successful):

1. Country/region code of the application server
2. Code page of the application server
3. Authorization ID
4. Node name (as specified on the API)
5. Identity and platform type of the server
6. Agent ID of the agent which has been started at the server
7. Agent index
8. Node number of the server
9. Number of database partitions if the server is a partitioned database server.

If the node name is a zero-length string or NULL, information about the current state of attachment is returned. If no attachment exists, sqlcode 1427 is returned. Otherwise, information about the attachment is returned in the sqlerrmc field of the sqlca (as outlined above).

If an attachment has not been made, instance-level APIs are executed against the current instance, specified by the DB2INSTANCE environment variable.

Certain functions (db2start, db2stop, and all directory services, for example) are never executed remotely. That is, they affect only the local instance environment, as defined by the value of the DB2INSTANCE environment variable.

If an attachment exists, and the API is issued with a node name, the current attachment is dropped, and an attachment to the new node is attempted.

Where the user name and password are authenticated depends on the authentication type of the target instance.

The node to which an attachment is to be made can also be specified by a call to the sqlesetc API.

REXX API syntax:

```
ATTACH [TO nodename [USER username USING password]]
```

REXX API parameters:

nodename
Alias of the instance to which the user wants to attach. This instance must have a matching entry in the local node directory. The only exception is the local instance (as specified by the DB2INSTANCE environment variable), which can be specified as the object of an attachment, but cannot be used as a node name in the node directory.

username
Name under which the user attaches to the instance.

password
Password used to authenticate the user name.

Related tasks:

- "Attaching to and detaching from a non-default instance of the database manager" in *Administration Guide: Implementation*

Related reference:
- "ATTACH " on page 631
- "SQLCA data structure" in *Administrative API Reference*
- "sqleatcp - Attach to instance and change password" on page 836
- "sqledtin - Detach from instance" on page 841
- "sqlesetc API - Set client connection settings" in *Administrative API Reference*

Related samples:
- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

sqledtin - Detach from instance

Removes the logical instance attachment, and terminates the physical communication connection if there are no other logical connections using this layer.

Authorization:

None

Required connection:

None. Removes an existing instance attachment.

API include file:

```
sqlenv.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  sqledtin (
        struct sqlca * pSqlca);

SQL_API_RC SQL_API_FN
  sqlgdtin (
        struct sqlca * pSqlca);
```

sqledtin API parameters:

pSqlca
Output. A pointer to the sqlca structure.

REXX API syntax:

```
DETACH
```

Related tasks:
- "Attaching to and detaching from a non-default instance of the database manager" in *Administration Guide: Implementation*

Related reference:
- "DETACH " on page 633

- "sqleatin - Attach to instance" on page 838
- "SQLCA data structure" in *Administrative API Reference*
- "sqleatcp - Attach to instance and change password" on page 836

Related samples:
- "dbinst.cbl -- Attach to and detach from an instance (IBM COBOL)"
- "inattach.c -- Attach to and detach from an instance (C)"
- "utilapi.c -- Error-checking utility for non-embedded SQL samples in C (C)"
- "inattach.C -- Attach to and detach from an instance (C++)"
- "utilapi.C -- Checks for and prints to the screen SQL warnings and errors (C++)"

Part 11. SQL statements

Chapter 40. SQL Statements for Administrators

ALTER DATABASE PARTITION GROUP

The ALTER DATABASE PARTITION GROUP statement is used to:

- add one or more database partitions to a database partition group
- drop one or more database partitions from a database partition group.

Invocation:

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The authorization ID of the statement must have SYSCTRL or SYSADM authority.

Syntax:

```
►►──ALTER DATABASE PARTITION GROUP──db-partition-name──────────────────────────────────►

     ┌─,───────────────────────────────────────────────────────────────────────────┐
     ▼                                                                             │
►────┬─ADD──┬─DBPARTITIONNUM──┬─┤ db-partitions-clause ├─┬────────────────────────────────────────┬─┬┴──►◄
     │      └─DBPARTITIONNUMS─┘                          ├─LIKE DBPARTITIONNUM──db-partition-number─┤ │
     │                                                   └─WITHOUT TABLESPACES─────────────────────┘ │
     └─DROP──┬─DBPARTITIONNUM──┬─┤ db-partitions-clause ├────────────────────────────────────────────┘
             └─DBPARTITIONNUMS─┘
```

db-partitions-clause:

```
      ┌─,──────────────────────────────────────────────┐
      ▼                                                │
├──(────db-partition-number1──┬─────────────────────────┬┴──)──────────────┤
                               └─TO──db-partition-number2─┘
```

Description:

db-partition-name
Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). It must be a database partition group described in the catalog. IBMCATGROUP and IBMTEMPGROUP cannot be specified (SQLSTATE 42832).

ADD DBPARTITIONNUM
Specifies the specific database partition or partitions to add to the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must not already be defined in the database partition group (SQLSTATE 42728).

DROP DBPARTITIONNUM
Specifies the specific database partition or partitions to drop from the database partition group. DBPARTITIONNUMS is a synonym for DBPARTITIONNUM. Any specified database partition must already be defined in the database partition group (SQLSTATE 42729).

db-partitions-clause
Specifies the database partition or partitions to be added or dropped.

db-partition-number1
Specify a specific database partition number.

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9).

LIKE DBPARTITIONNUM *db-partition-number*
Specifies that the containers for the existing table spaces in the database partition group will be the same as the containers on the specified *db-partition-number*. The specified database partition must be a partition that existed in the database partition group prior to this statement, and that is not included in a DROP DBPARTITIONNUM clause of the same statement.

For table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all), the containers will not necessarily match those from the specified partition. Instead, containers will automatically be assigned by the database manager based on the storage paths that are associated with the database, and this might or might not result in the same containers being used. The size of each table space is based on the initial size that was specified when the table space was created, and might not match the current size of the table space on the specified partition.

WITHOUT TABLESPACES
Specifies that the containers for existing table spaces in the database partition group are not created on the newly added database partition or partitions. The ALTER TABLESPACE statement using the FOR DBPARTITIONNUM clause must be used to define containers for use with the table spaces that are defined

on this database partition group. If this option is not specified, the default containers are specified on newly added database partitions for each table space defined on the database partition group.

This option is ignored for table spaces that are defined to use automatic storage (that is, table spaces that were created with the MANAGED BY AUTOMATIC STORAGE clause of the CREATE TABLESPACE statement, or for which no MANAGED BY clause was specified at all). There is no way to defer container creation for these table spaces. Containers will automatically be assigned by the database manager based on the storage paths that are associated with the database. The size of each table space will be based on the initial size that was specified when the table space was created.

Rules:

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the ON DBPARTITIONNUMS clause must be for a unique database partition (SQLSTATE 42728).
- A valid database partition number is between 0 and 999 inclusive (SQLSTATE 42729).
- A database partition cannot appear in both the ADD and DROP clauses (SQLSTATE 42728).
- There must be at least one database partition remaining in the database partition group. The last database partition cannot be dropped from a database partition group (SQLSTATE 428C0).
- If neither the LIKE DBPARTITIONNUM clause nor the WITHOUT TABLESPACES clause is specified when adding a database partition, the default is to use the lowest database partition number of the existing database partitions in the database partition group (say it is 2) and proceed as if LIKE DBPARTITIONNUM 2 had been specified. For an existing database partition to be used as the default, it must have containers defined for all the table spaces in the database partition group (column IN_USE of SYSCAT.DBPARTITIONGROUPDEF is not ’T’).

Notes:

- When a database partition is added to a database partition group, a catalog entry is made for the database partition (see SYSCAT.DBPARTITIONGROUPDEF). The distribution map is changed immediately to include the new database partition, along with an indicator (IN_USE) that the database partition is in the distribution map if either:
  - no table spaces are defined in the database partition group or
  - no tables are defined in the table spaces defined in the database partition group and the WITHOUT TABLESPACES clause was not specified.

  The distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is not included in the distribution map if either:
  - Tables exist in table spaces in the database partition group or
  - Table spaces exist in the database partition group and the WITHOUT TABLESPACES clause was specified (unless all of the table spaces are defined to use automatic storage, in which case the WITHOUT TABLESPACES clause is ignored)

  To change the distribution map, the REDISTRIBUTE DATABASE PARTITION GROUP command must be used. This redistributes any data, changes the distribution map, and changes the indicator. Table space containers need to be added before attempting to redistribute data if the WITHOUT TABLESPACES clause was specified.

- When a database partition is dropped from a database partition group, the catalog entry for the database partition (see SYSCAT.DBPARTITIONGROUPDEF) is updated. If there are no tables defined in the table spaces defined in the database partition group, the distribution map is changed immediately to exclude the dropped database partition and the entry for the database partition in the database partition group is dropped. If tables exist, the distribution map is not changed and the indicator (IN_USE) is set to indicate that the database partition is waiting to be dropped. The REDISTRIBUTE DATABASE PARTITION GROUP command must be used to redistribute the data and drop the entry for the database partition from the database partition group.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
    - NODES can be specified in place of DBPARTITIONNUMS
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Example:

Assume that you have a six-partition database that has the following database partitions: 0, 1, 2, 5, 7, and 8. Two database partitions (3 and 6) are added to the system.

- Assume that you want to add database partitions 3 and 6 to a database partition group called MAXGROUP, and have table space containers like those on database partition 2. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MAXGROUP
  ADD DBPARTITIONNUMS (3,6)LIKE DBPARTITIONNUM 2
```

- Assume that you want to drop database partition 1 and add database partition 6 to database partition group MEDGROUP. You will define the table space containers separately for database partition 6 using ALTER TABLESPACE. The statement is as follows:

```
ALTER DATABASE PARTITION GROUP MEDGROUP
  ADD DBPARTITIONNUM(6)WITHOUT TABLESPACES
  DROP DBPARTITIONNUM(1)
```

Related concepts:

- Chapter 9, “Database partitioning across multiple database partitions,” on page 49
- “Automatic storage databases” in *Administration Guide: Implementation*

Related reference:

- “sqleaddn - Add a database partition server to the partitioned database environment” on page 792

ALTER FUNCTION

The ALTER FUNCTION statement modifies the properties of an existing function.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- ALTERIN privilege on the schema of the function
- Definer of the function, as recorded in the DEFINER column of the SYSCAT.ROUTINES catalog view
- SYSADM or DBADM authority

To alter the EXTERNAL NAME of a function, the privileges held by the authorization ID of the statement must also include at least one of the following:
- CREATE_EXTERNAL_ROUTINE authority on the database
- SYSADM or DBADM authority

To alter a function to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following:
- CREATE_NOT_FENCED_ROUTINE authority on the database
- SYSADM or DBADM authority

To alter a function to be fenced, no additional authorities or privileges are required.

Syntax:

```
                                      ┌──────────────────────────────────────────┐
                                      ▼                                          │
►►──ALTER──┤ function-designator ├──┬─EXTERNAL NAME──┬─'string'───┬─┬─┴──────────►◄
                                    │                └─identifier─┘ │
                                    ├─┬─FENCED─────┬────────────────┤
                                    │ └─NOT FENCED─┘                │
                                    └─┬─THREADSAFE─────┬────────────┘
                                      └─NOT THREADSAFE─┘
```

Description:

function-designator
: Uniquely identifies the function to be altered. For more information, see "Function, method, and procedure designators" on page 1286.

EXTERNAL NAME *'string'* **or** *identifier*
: Identifies the name of the user-written code that implements the function. This option can only be specified when altering external functions (SQLSTATE 42849).

FENCED or **NOT FENCED**
: Specifies whether the function is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most functions have the option of running as FENCED or NOT FENCED.

If a function is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the function. In general, a function running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for functions that were not adequately coded, reviewed, and tested can compromise the integrity of DB2. DB2 takes some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED user-defined functions are used.

A function declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a function has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the function cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE, OLEDB, or CLR functions (SQLSTATE 42849).

THREADSAFE or **NOT THREADSAFE**
Specifies whether the function is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the function is defined with LANGUAGE other than OLE and OLEDB:

- If the function is defined as THREADSAFE, the database manager can invoke the function in the same process as other routines. In general, to be threadsafe, a function should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED functions can be THREADSAFE.
- If the function is defined as NOT THREADSAFE, the database manager will never simultaneously invoke the function in the same process as another routine. Only a fenced function can be NOT THREADSAFE (SQLSTATE 42613).

This option may not be altered for LANGUAGE OLE or OLEDB functions (SQLSTATE 42849).

Notes:

- It is not possible to alter a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Functions declared as LANGUAGE SQL, sourced functions, or template functions cannot be altered (SQLSTATE 42917).

Example:

The function MAIL() has been thoroughly tested. To improve its performance, alter the function to be not fenced.

```
ALTER FUNCTION MAIL() NOT FENCED
```

Related reference:

- "Function, method, and procedure designators" on page 1286
- "CREATE FUNCTION (External Scalar) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Table) statement" in *SQL Reference, Volume 2*

- “CREATE FUNCTION (OLE DB External Table) statement” in *SQL Reference, Volume 2*

ALTER METHOD

The ALTER METHOD statement modifies an existing method by changing the method body associated with the method.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- CREATE_EXTERNAL_ROUTINE authority on the database, and at least one of:
  - ALTERIN privilege on the schema of the type
  - Definer of the type, as recorded in the DEFINER column of the SYSCAT.DATATYPES catalog view
- SYSADM or DBADM authority

Syntax:

```
►►──ALTER──┤ method-designator ├──EXTERNAL NAME──┬─'string'───┬──────────────────►◄
                                                 └─identifier─┘
```

Description:

method-designator
: Uniquely identifies the method to be altered. For more information, see “Function, method, and procedure designators” on page 1286.

EXTERNAL NAME *'string'* **or** *identifier*
: Identifies the name of the user-written code that implements the method. This option can only be specified when altering external methods (SQLSTATE 42849).

Notes:
- It is not possible to alter a method that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).
- Methods declared as LANGUAGE SQL cannot be altered (SQLSTATE 42917).
- Methods declared as LANGUAGE CLR cannot be altered (SQLSTATE 42849).
- The specified method must have a body before it can be altered (SQLSTATE 42704).

Example:

Alter the method DISTANCE() in the structured type ADDRESS_T to use the library newaddresslib.

```
ALTER METHOD DISTANCE()
  FOR TYPE ADDRESS_T
  EXTERNAL NAME 'newaddresslib!distance2'
```

Related reference:

- "Function, method, and procedure designators" on page 1286
- "CREATE METHOD " on page 937

ALTER PROCEDURE

The ALTER PROCEDURE statement modifies an existing procedure by changing the properties of the procedure.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- ALTERIN privilege on the schema of the procedure
- Definer of the procedure, as recorded in the DEFINER column of the SYSCAT.ROUTINES catalog view
- SYSADM or DBADM authority

To alter the EXTERNAL NAME of a procedure, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_EXTERNAL_ROUTINE authority on the database
- SYSADM or DBADM authority

To alter a procedure to be not fenced, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_NOT_FENCED_ROUTINE authority on the database
- SYSADM or DBADM authority

To alter a procedure to be fenced, no additional authorities or privileges are required.

Syntax:

```
                                          ┌──────────────────────────────────────┐
                                          ▼                                      │
►►──ALTER──┤ procedure-designator ├──┬──EXTERNAL NAME──┬─'string'───┬───┴──────►◄
                                      │                 └─identifier─┘   │
                                      ├──┬─FENCED─────┬──────────────────┤
                                      │  └─NOT FENCED─┘                  │
                                      ├──┬─EXTERNAL ACTION────┬──────────┤
                                      │  └─NO EXTERNAL ACTION─┘          │
                                      ├──┬─THREADSAFE─────┬──────────────┤
                                      │  └─NOT THREADSAFE─┘              │
                                      └──NEW SAVEPOINT LEVEL─────────────┘
```

Description:

procedure-designator
Uniquely identifies the procedure to be altered. A sourced procedure cannot be specified (SQLSTATE 25000). For more information, see "Function, method, and procedure designators" on page 1286.

EXTERNAL NAME *'string'* **or** *identifier*
Identifies the name of the user-written code that implements the procedure. This option can only be specified when altering external procedures (SQLSTATE 42849). The EXTERNAL NAME clause cannot be altered in procedures that were declared as LANGUAGE SQL (SQLSTATE 42917).

FENCED or **NOT FENCED**
Specifies whether the procedure is considered safe to run in the database manager operating environment's process or address space (NOT FENCED), or not (FENCED). Most procedures have the option of running as FENCED or NOT FENCED.

If a procedure is altered to be FENCED, the database manager insulates its internal resources (for example, data buffers) from access by the procedure. In general, a procedure running as FENCED will not perform as well as a similar one running as NOT FENCED.

CAUTION:
Use of NOT FENCED for procedures that were not adequately coded, reviewed, and tested can compromise the integrity of DB2. DB2 takes some precautions against many of the common types of inadvertent failures that might occur, but cannot guarantee complete integrity when NOT FENCED stored procedures are used.

This option can only be specified when altering external procedures (SQLSTATE 42849).

A procedure declared as NOT THREADSAFE cannot be altered to be NOT FENCED (SQLSTATE 42613).

If a procedure has any parameters defined AS LOCATOR, and was defined with the NO SQL option, the procedure cannot be altered to be FENCED (SQLSTATE 42613).

This option cannot be altered for LANGUAGE OLE or CLR procedures (SQLSTATE 42849).

The FENCED or NOT FENCED clause cannot be altered in procedures that were declared as LANGUAGE SQL (SQLSTATE 42917).

EXTERNAL ACTION or **NO EXTERNAL ACTION**
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not

(NO EXTERNAL ACTION). If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

THREADSAFE or **NOT THREADSAFE**

Specifies whether the procedure is considered safe to run in the same process as other routines (THREADSAFE), or not (NOT THREADSAFE).

If the procedure is defined with LANGUAGE other than OLE:

- If the procedure is defined as THREADSAFE, the database manager can invoke the procedure in the same process as other routines. In general, to be threadsafe, a procedure should not use any global or static data areas. Most programming references include a discussion of writing threadsafe routines. Both FENCED and NOT FENCED procedures can be THREADSAFE.
- If the procedure is defined as NOT THREADSAFE, the database manager will never invoke the procedure in the same process as another routine. Only a fenced procedure can be NOT THREADSAFE (SQLSTATE 42613).

This option can only be specified when altering external procedures (SQLSTATE 42849).

This option cannot be altered for LANGUAGE OLE procedures (SQLSTATE 42849).

The THREADSAFE or NOT THREADSAFE clause cannot be altered in procedures that were declared as LANGUAGE SQL (SQLSTATE 42917).

NEW SAVEPOINT LEVEL

Specifies that a new savepoint level is to be created for the procedure. A savepoint level refers to the scope of reference for any savepoint-related statement, as well as to the name space used for comparison and reference of any savepoint names.

The savepoint level for a procedure can only be altered to NEW SAVEPOINT LEVEL.

Rules:

- It is not possible to alter a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

Example:

Alter the procedure PARTS_ON_HAND() to be not fenced.

```
ALTER PROCEDURE PARTS_ON_HAND() NOT FENCED
```

Related reference:

- "Function, method, and procedure designators" on page 1286
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*

ALTER TABLE

The ALTER TABLE statement alters the definition of a table.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- ALTER privilege on the table to be altered
- CONTROL privilege on the table to be altered
- ALTERIN privilege on the schema of the table
- SYSADM or DBADM authority

To create or drop a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:

- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- SYSADM or DBADM authority

To drop the primary key or a unique constraint on table T, the privileges held by the authorization ID of the statement must include at least one of the following on every table that is a dependent of this parent key of T:

- ALTER privilege on the table
- CONTROL privilege on the table
- ALTERIN privilege on the schema of the table
- SYSADM or DBADM authority

To alter a table to become a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the table
- SYSADM or DBADM authority

and at least one of the following, on each table or view identified in the fullselect:

- SELECT and ALTER privilege on the table or view
- CONTROL privilege on the table or view
- SELECT privilege on the table or view, and ALTERIN privilege on the schema of the table or view
- SYSADM or DBADM authority

To alter a table so that it is no longer a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following, on each table or view identified in the fullselect used to define the materialized query table:

- ALTER privilege on the table or view
- CONTROL privilege on the table or view
- ALTERIN privilege on the schema of the table or view
- SYSADM or DBADM authority

To add a column of type DB2SECURITYLABEL to a table, the privileges held by the authorization ID of the statement must include at least a security label from the security policy associated with the table.

To remove the security policy from a table, the privileges held by the authorization ID of the statement must include SECADM authority.

To alter a table to attach a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following on the source table:

- SELECT privilege on the table and DROPIN privilege on the schema of the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

and at least one of the following on the target table:

- ALTER and INSERT privileges on the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

To alter a table to detach a data partition, the privileges held by the authorization ID of the statement must also include at least one of the following on the target table of the detached partition:

- CREATETAB authority on the database, and USE privilege on the table spaces used by the table, as well as one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the new table does not exist
  - CREATEIN privilege on the schema, if the schema name of the new table refers to an existing schema
- SYSADM or DBADM authority

and at least one of the following on the source table:

- SELECT, ALTER, and DELETE privileges on the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

Syntax:

```
►►──ALTER TABLE──table-name──────────────────────────────────────────────►
```

```
          .-----------------------------------------------------------------------.
          V                                                                       |
>>--+--+----ADD--+--------+--| column-definition |-----------------+--------------+--+--------><
    |  |         '-COLUMN-'                                        |              |  |
    |  |         +--| unique-constraint |------------------------+             |  |
    |  |         +--| referential-constraint |-------------------+             |  |
    |  |         +--| check-constraint |-------------------------+             |  |
    |  |         +--| distribution-clause |----------------------+             |  |
    |  |         '--RESTRICT ON DROP-------------------------------'             |  |
    |  |            .-MATERIALIZED-.                                             |  |
    |  +----ADD--+--+--------------+--QUERY-+--| materialized-query-definition |--+  |
    |  |         '----------------------------'                                  |  |
    |  +----ADD PARTITION--| add-partition |-------------------------------------+  |
    |  +----ALTER--+-FOREIGN KEY-+--constraint-name--| constraint-alteration |---+  |
    |  |           '-CHECK-------'                                               |  |
    |  |           .-COLUMN-.                                                    |  |
    |  +----ALTER--+--------+--| column-alteration |-----------------------------+  |
    |  +----ATTACH PARTITION--| attach-partition |--------------------------------+  |
    |  +----DETACH PARTITION--partition-name--INTO--table-name1-------------------+  |
    |  +----DROP--+-PRIMARY KEY--------------------------------------------------+  |
    |  |          +-+-FOREIGN KEY-+--constraint-name--------------+              |  |
    |  |          | +-UNIQUE------+                               |              |  |
    |  |          | +-CHECK-------+                               |              |  |
    |  |          | '-CONSTRAINT--'                               |              |  |
    |  |          |  .-COLUMN-.                 .-CASCADE--.      |              |  |
    |  |          +--+--------+--column-name--+-+----------+-+----+              |  |
    |  |          |                             '-RESTRICT-'      |              |  |
    |  |          +-PARTITIONING KEY------------------------------+              |  |
    |  |          '-RESTRICT ON DROP------------------------------'              |  |
    |  +----DROP DISTRIBUTION--------------------------------------------------------+  |
    |  |          .-MATERIALIZED-.                                               |  |
    |  +----DROP--+--------------+--QUERY----------------------------------------+  |
    |  +----DATA CAPTURE--+-NONE------------------------------------------+------+  |
    |  |                  '-CHANGES--+--------------------------------+---'      |  |
    |  |                             '-INCLUDE LONGVAR COLUMNS--------'          |  |
    |  +----ACTIVATE NOT LOGGED INITIALLY--+----------------------+---------------+  |
    |  |                                   '-WITH EMPTY TABLE-----'              |  |
    |  +----PCTFREE--integer------------------------------------------------------+  |
    |  +----LOCKSIZE--+-ROW---------+-----------------------------------------------+  |
    |  |              +-BLOCKINSERT-+                                            |  |
    |  |              '-TABLE-------'                                            |  |
    |  +----APPEND--+-ON--+------------------------------------------------------+  |
    |  |            '-OFF-'                                                      |  |
    |  |                          .-CARDINALITY-.                                |  |
    |  +--+-VOLATILE-----+--+-------------+--------------------------------------+  |
    |  |  '-NOT VOLATILE-'                                                       |  |
    |  +----COMPRESS--+-YES-+----------------------------------------------------+  |
    |  |              '-NO--'                                                    |  |
    |  +--+-ACTIVATE---+--VALUE COMPRESSION--------------------------------------+  |
    |  |  '-DEACTIVATE-'                                                         |  |
    |  '----LOG INDEX BUILD--+-NULL-+--------------------------------------------'  |
    |                        +-OFF--+                                               |
    |                        '-ON---'                                               |
    +----ADD SECURITY POLICY--policy-name---------------------------------------------+
    '----DROP SECURITY POLICY---------------------------------------------------------'
```

column-definition:

```
|--column-name--+-----------------------+--+--------------------------+--------------|
                |                   (1) |  '--| column-options |------'
                '--| data-type |--------'
```

column-options:

```
    .-----------------------------------------------------------------------------.
    V                                                                             |
|------+-NOT NULL----------------------------------------------------------------+-+---|
       |                  (2)                                                    |
       +-| lob-options |-------------------------------------------------------+
       |                       (3)                                               |
       +-| datalink-options |--------------------------------------------------+
       |                                  (4)                                    |
       +-SCOPE--+-typed-table-name2-+--------------------------------------------+
       |        '-typed-view-name2--'                                            |
       +-+------------------------------------+--+-+-PRIMARY KEY-+-------------+-+
       | '-CONSTRAINT--constraint-name--'        | '-UNIQUE------'             | |
       |                                         +-| references-clause |-------+ |
       |                                         '-CHECK--(--check-condition--)--| constraint-attributes |-' |
       +-| generated-column-spec |-----------------------------------------------+
       +-COMPRESS SYSTEM DEFAULT-------------------------------------------------+
       |  .-COLUMN-.                                                             |
       '--+--------+--SECURED WITH--security-label-name--------------------------'
```

lob-options:

```
        .-LOGGED-----.     .-NOT COMPACT-.
|--●--+------------+--●--+-------------+--●----------------------------------|
      '-NOT LOGGED-'     '-COMPACT-----'
```

datalink-options:

```
                    .-NO LINK CONTROL----------------------------------.
|--LINKTYPE URL--+--------------------------------------------------+-----------|
                 '-FILE LINK CONTROL--+-| file-link-options |-+-'
                                      '-MODE DB2OPTIONS-------'
```

file-link-options:

```
|--●--INTEGRITY--ALL--●--READ PERMISSION--+-FS-+------------------------------->
                                          '-DB-'

>--●--WRITE PERMISSION--+-FS----------------------------------------------+---->
                        +-BLOCKED-----------------------------------------+
                        '-ADMIN--+-----+--REQUIRING TOKEN FOR UPDATE------'
                                 '-NOT-'

>--●--RECOVERY--+-NO--+--●--ON UNLINK--+-RESTORE-+--●-------------------------|
                '-YES-'                '-DELETE--'
```

references-clause:

```
|--REFERENCES--+-table-name-+--+-----------------------+---------------------->
               '-nickname---'  |    .-,-----------.    |
                               |    V             |    |
                               '-(----column-name-+--)-'

>--| rule-clause |--| constraint-attributes |----------------------------------|
```

rule-clause:

```
      ┌─ON DELETE NO ACTION───────────┐   ┌─ON UPDATE NO ACTION─┐
├──●──┼───────────────────────────────┼─●─┼─────────────────────┼──●────────────┤
      └─ON DELETE──┬─RESTRICT─┬───────┘   └─ON UPDATE RESTRICT──┘
                   ├─CASCADE──┤
                   └─SET NULL─┘
```

constraint-attributes:

```
      ┌─ENFORCED─────┐   ┌─ENABLE QUERY OPTIMIZATION──┐
├──●──┼──────────────┼─●─┼────────────────────────────┼──●────────────┤
      └─NOT ENFORCED─┘   └─DISABLE QUERY OPTIMIZATION─┘
```

generated-column-spec:

```
├──┬──┤ default-clause ├────────────────────────────────────────┬────────┤
   │            ┌─ALWAYS─────┐                          (5)     │
   ├─GENERATED──┼────────────┼──┤ identity-options ├────────────┤
   │            └─BY DEFAULT─┘                                  │
   │            ┌─ALWAYS─┐                                      │
   └─GENERATED──┴────────┴──AS──(──generation-expression──)─────┘
```

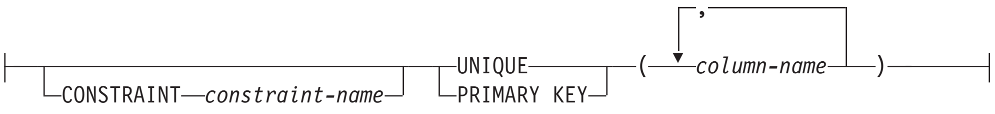
default-clause:

```
    ┌─WITH─┐
├───┴──────┴──DEFAULT──┬─────────────────────────────────────────────────────┬──────┤
                       ├─constant────────────────────────────────────────────┤
                       ├─datetime-special-register───────────────────────────┤
                       ├─user-special-register───────────────────────────────┤
                       ├─CURRENT SCHEMA──────────────────────────────────────┤
                       ├─NULL────────────────────────────────────────────────┤
                       └─cast-function──(──┬─constant──────────────────┬──)──┘
                                           ├─datetime-special-register─┤
                                           ├─user-special-register─────┤
                                           └─CURRENT SCHEMA────────────┘
```

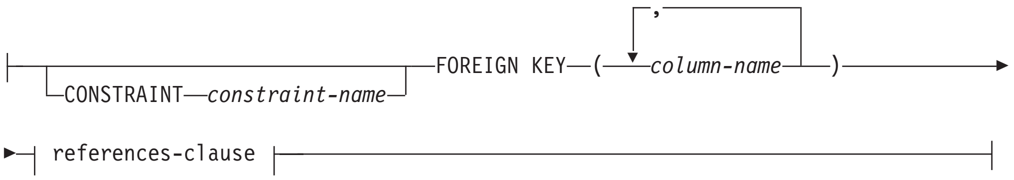
identity-options:

```
├──AS IDENTITY──┬──────────────────────────────────────────────────────────────┬──────┤
                │    ┌──────────────────────────────────────────────────┐      │
                │    ▼    (6)                   ┌─1────────────────┐    │      │
                └─(──────────┬─START WITH───────┴─numeric-constant─┴──┬─┴──)───┘
                             │                  ┌─1────────────────┐  │
                             ├─INCREMENT BY─────┴─numeric-constant─┴──┤
                             │  ┌─NO MINVALUE──────────────────┐      │
                             ├──┴─MINVALUE──numeric-constant─┴──────┤
                             │  ┌─NO MAXVALUE──────────────────┐      │
                             ├──┴─MAXVALUE──numeric-constant─┴──────┤
                             │  ┌─NO CYCLE─┐                          │
                             ├──┴─CYCLE────┴──────────────────────────┤
                             │  ┌─CACHE 20──────────────────┐         │
                             └──┼─NO CACHE──────────────────┼─────────┘
                                └─CACHE──integer-constant───┘
```

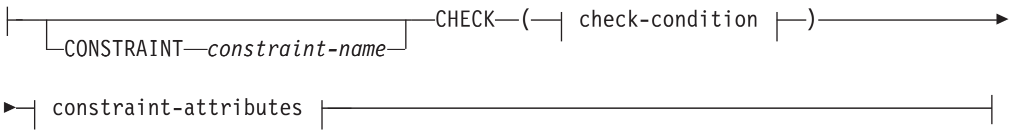
unique-constraint:



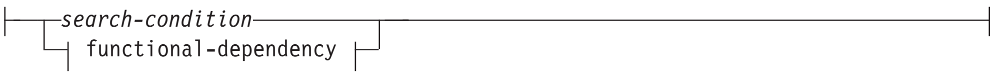
referential-constraint:



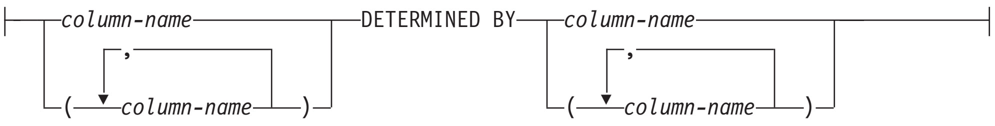
check-constraint:



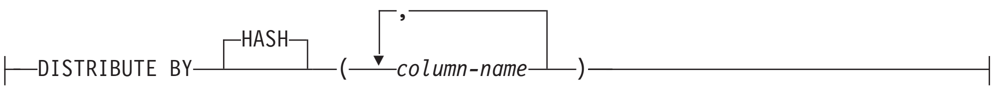
check-condition:



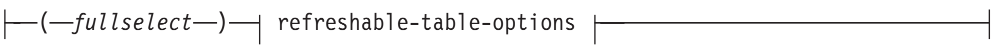
functional-dependency:



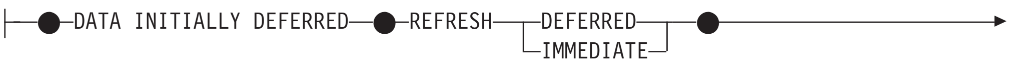
distribution-clause:

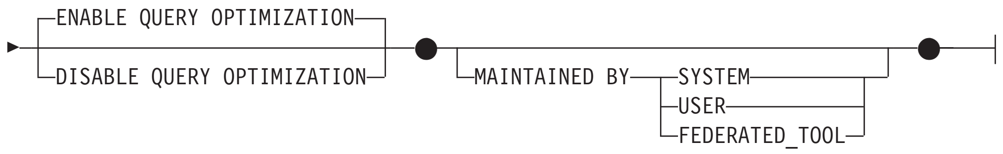


materialized-query-definition:

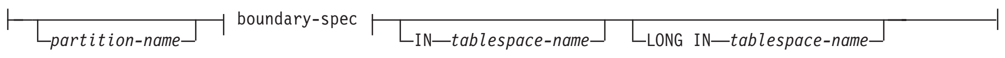


refreshable-table-options:

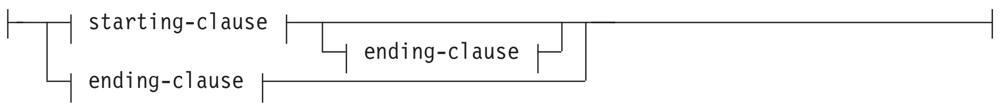




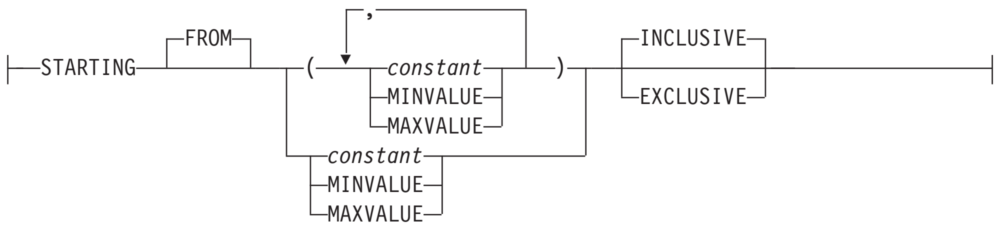
add-partition:



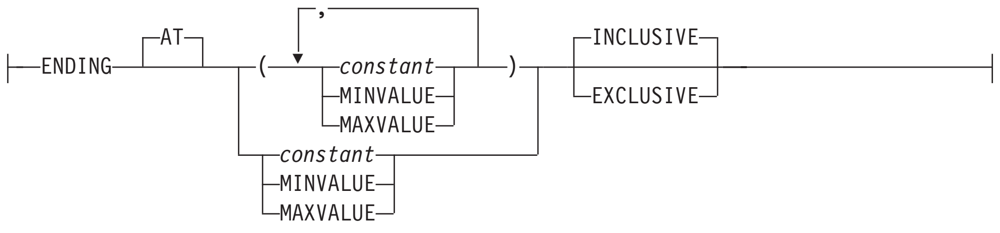
boundary-spec:



starting-clause:



ending-clause:



duration-label:

```
├──┬─YEAR─────────┬──────────────────────────────────────────────────────────┤
   ├─YEARS────────┤
   ├─MONTH────────┤
   ├─MONTHS───────┤
   ├─DAY──────────┤
   ├─DAYS─────────┤
   ├─HOUR─────────┤
   ├─HOURS────────┤
   ├─MINUTE───────┤
   ├─MINUTES──────┤
   ├─SECOND───────┤
   ├─SECONDS──────┤
   ├─MICROSECOND──┤
   └─MICROSECONDS─┘
```

constraint-alteration:

```
   ┌─────────────────────────────────────────────┐
   ▼    (6)                                      │
├──────────┬─┬─ENABLE──┬──QUERY OPTIMIZATION─┬──┴──────────────────────────┤
           │ └─DISABLE─┘                     │
           └─┬─────┬──ENFORCED───────────────┘
             └─NOT─┘
```

column-alteration:

```
├──column-name──┬─SET──┬─DATA TYPE──┤ altered-data-type ├──────┬──────┬──┤
                │      ├─┤ generated-column-spc ├──────────────┤      │
                │      ├─EXPRESSION AS──(──generation-expression──)─┤ │
                │      ├─INLINE LENGTH──integer────────────────┤      │
                │      └─NOT NULL──────────────────────────────┘      │
                ├─┬─┤ generation-alteration ├─┬──────────────────────┬─┤
                │ │                           └─┤ identity-alteration ├┘│
                │ └─┤ identity-alteration ├──────────────────────────┘ │
                ├─DROP──┬─IDENTITY───┬─────────────────────────────────┤
                │       ├─EXPRESSION─┤                                 │
                │       ├─DEFAULT────┤                                 │
                │       └─NOT NULL───┘                                 │
                ├─ADD SCOPE──┬─typed-table-name─┬──────────────────────┤
                │            └─typed-view-name──┘                      │
                ├─COMPRESS──┬─SYSTEM DEFAULT─┬─────────────────────────┤
                │           └─OFF────────────┘                         │
                ├─SECURED WITH──security-label-name────────────────────┤
                └─DROP COLUMN SECURITY─────────────────────────────────┘
```

altered-data-type:

```
├──┬─┬─INTEGER─┬───────────────────────────────────────────────────┬──────────┤
   │ └─INT─────┘                                                   │
   ├─BIGINT────────────────────────────────────────────────────────┤
   ├─┬─FLOAT─┬───────────────┬─┬───────────────────────────────────┤
   │ │       └─(─integer─)───┘ │                                   │
   │ ├─REAL────────────────────┤                                   │
   │ │        ┌─PRECISION─┐    │                                   │
   │ └─DOUBLE─┴───────────┴────┘                                   │
   ├─┬─DECIMAL─┬─┬─────────────────────────────────┬───────────────┤
   │ ├─DEC─────┤ └─(─integer─┬────────────┬─)─────┘               │
   │ ├─NUMERIC─┤             └─,integer───┘                        │
   │ └─NUM─────┘                                                   │
   ├─┬─┬─CHARACTER─┬─┬──────────────┬──┬─┬──────────────────┬──────┤
   │ │ └─CHAR──────┘ └─(─integer─)──┘  │ └─FOR BIT DATA─────┘      │
   │ └─┬─VARCHAR───────────────────┬─(─integer─)─┘                 │
   │   └─┬─CHARACTER─┬─VARYING─────┘                               │
   │     └─CHAR──────┘                                             │
   ├─┬─┬─BLOB────────────────┬──────────┬─┬──────────────────────┬─┤
   │ │ └─BINARY LARGE OBJECT─┘          │ └─(─integer─┬───┬─)────┘ │
   │ ├─┬─CLOB────────────────────────┬──┤             ├─K─┤        │
   │ │ └─┬─CHARACTER─┬─LARGE OBJECT──┘  │             ├─M─┤        │
   │ │   └─CHAR──────┘                  │             └─G─┘        │
   │ └─DBCLOB───────────────────────────┘                          │
   ├─GRAPHIC─┬───────────────┬─────────────────────────────────────┤
   │         └─(─integer─)───┘                                     │
   └─VARGRAPHIC─(─integer─)────────────────────────────────────────┘
```

generation-alteration:

```
├──SET GENERATED──┬─ALWAYS─────┬──────────────────────────────────────────────┤
                  └─BY DEFAULT─┘
```

identity-alteration:

```
   ┌──────────────────────────────────────────────────┐
   ↓ (6)                                              │
├──────────┬─SET INCREMENT BY─numeric-constant──────┬─┴─────────────────────────┤
           ├─SET──┬─NO MINVALUE──────────────────┬──┤
           │      └─MINVALUE─numeric-constant────┘  │
           ├─SET──┬─NO MAXVALUE──────────────────┬──┤
           │      └─MAXVALUE─numeric-constant────┘  │
           ├─SET──┬─NO CYCLE─┬──────────────────────┤
           │      └─CYCLE────┘                      │
           ├─SET──┬─NO CACHE──────────────────┬─────┤
           │      └─CACHE─integer-constant────┘     │
           ├─SET──┬─NO ORDER─┬──────────────────────┤
           │      └─ORDER────┘                      │
           └─RESTART─┬─────────────────────────┬────┘
                     └─WITH─numeric-constant───┘
```

attach-partition:

```
├──┬──────────────────┬──┤ boundary-spec ├──FROM─table-name──────────────────┤
   └─partition-name───┘
```

Notes:

1 If the first column option chosen is *generated-column-spec*, *data-type* can be omitted; it will be computed by the generation expression.

2 The *lob-options* clause only applies to large object types (BLOB, CLOB and DBCLOB), and to distinct types that are based on large object types.

3 The *datalink-options* clause only applies to the DATALINK type and to distinct types that are based on the DATALINK type.

4 The SCOPE clause only applies to the REF type.

5 Identity options cannot be specified for *column-definition*.

6 The same clause must not be specified more than once.

Description:

table-name
The *table-name* must identify a table that exists at the current server. It cannot be a nickname (SQLSTATE 42809) and must not be a view, a catalog table, or a declared temporary table (SQLSTATE 42995).

If *table-name* identifies a materialized query table, alterations are limited to adding or dropping the materialized query table, activating not logged initially, adding or dropping RESTRICT ON DROP, and changing pctfree, locksize, append, or volatile.

If *table-name* identifies a range-clustered table, alterations are limited to adding, changing, or dropping constraints, activating not logged initially, adding or dropping RESTRICT ON DROP, changing locksize, data capture, or volatile, and setting column default values.

ADD SECURITY POLICY *policy-name*
Adds a security policy to the table. The security policy must exist at the current server (SQLSTATE 42704). The table must not already have a security policy (SQLSTATE 55065), and must not be a typed table (SQLSTATE 428DH), materialized query table (MQT), or staging table (SQLSTATE 428FG).

DROP SECURITY POLICY
Removes the security policy and all LBAC protection from the table. The table specified by *table-name* must be protected by a security policy (SQLSTATE 428GT). If the table has a column with data type DB2SECURITYLABEL, the data type is changed to VARCHAR (128) FOR BIT DATA. If the table has one or more protected columns, those columns become unprotected.

ADD *column-definition*
Adds a column to the table. The table must not be a typed table (SQLSTATE 428DH). For all existing rows in the table, the value of the new column is set to its default value. The new column is the last column of the table; that is, if initially there are *n* columns, the added column is column *n*+1.

Adding the new column must not make the total byte count of all columns exceed the maximum record size.

column-name
Is the name of the column to be added to the table. The name cannot be qualified. Existing column names in the table cannot be used (SQLSTATE 42711).

data-type
Is one of the data types listed under "CREATE TABLE".

NOT NULL
Prevents the column from containing null values. The *default-clause* must also be specified (SQLSTATE 42601).

lob-options
Specifies options for LOB data types. See *lob-options* in “CREATE TABLE”.

datalink-options
Specifies options for DATALINK data types. See *datalink-options* in “CREATE TABLE”.

SCOPE
Specify a scope for a reference type column.

typed-table-name2
The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the value actually references an existing row in *typed-table-name2*.

typed-view-name2
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name2* (SQLSTATE 428DM). No checking is done of the default value for *column-name* to ensure that the values actually references an existing row in *typed-view-name2*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same ALTER TABLE statement, or as the name of any other existing constraint on the table (SQLSTATE 42710).

If the constraint name is not specified by the user, an 18 byte long identifier unique within the identifiers of the existing constraints defined on the table is generated by the system. (The identifier consists of ″SQL″ followed by a sequence of 15 numeric characters that are generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint. See “Notes” on page 890 for details on index names associated with unique constraints.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause were specified as a separate clause. The column cannot contain null values, so the NOT NULL attribute must also be specified (SQLSTATE 42831).

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause were specified as a separate clause.

See UNIQUE within the description of the *unique-constraint* below.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* in "CREATE TABLE".

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See *check-condition* in "CREATE TABLE".

generated-column-spec
For details on column generation, see "CREATE TABLE".

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a specific default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in Table 134. If a column is defined as a DATALINK, XML, or structured type, then a DEFAULT clause cannot be specified.

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

Table 134. Default Values (when no value specified)

Data Type	Default Value
Numeric	0
Fixed-length character string	Blanks
Varying-length character string	A string of length 0
Fixed-length graphic string	Double-byte blanks
Varying-length graphic string	A string of length 0
Date	For existing rows, a date corresponding to January 1, 0001. For added rows, the current date.
Time	For existing rows, a time corresponding to 0 hours, 0 minutes, and 0 seconds. For added rows, the current time.
Timestamp	For existing rows, a date corresponding to January 1, 0001, and a time corresponding to 0 hours, 0 minutes, 0 seconds and 0 microseconds. For added rows, the current timestamp.
Binary string (blob)	A string of length 0

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column.

Specific types of values that can be specified with the DEFAULT keyword are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified). For existing rows, the value is the current date, current time or current timestamp when the ALTER TABLE statement is processed.

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER. For existing rows, the value is the CURRENT USER, SESSION_USER, or SYSTEM_USER of the ALTER TABLE statement.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. For existing rows, the value of the CURRENT SCHEMA special register at the time the ALTER TABLE statement is processed.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL must not be specified within the same column definition.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the cast-function is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

If the value specified is not valid, an error (SQLSTATE 42894) is returned.

GENERATED
Specifies that DB2 generates values for the column.

ALWAYS
Specifies that DB2 will always generate a value for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* might change. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended option unless data propagation or unload and reload operations are being performed. GENERATED ALWAYS is the required option for generated columns.

BY DEFAULT
Specifies that DB2 will generate a value for the column when a row is inserted into the table, or updated, specifying DEFAULT for the column, unless an explicit value is specified. BY DEFAULT is the recommended option when using data propagation or performing unload and reload operations.

identity-options
This clause cannot be specified when adding a column to an existing table.

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. Requires that the table be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option. After the ALTER TABLE statement, the SET INTEGRITY statement with the IMMEDIATE CHECKED and FORCE GENERATED options must be used to update and check all the values in that column against the new expression. For details on specifying a column with a *generation-expression*, see "CREATE TABLE".

COMPRESS SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648) and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be a DATE, TIME, TIMESTAMP, XML, or structured data type (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

COLUMN SECURED WITH *security-label-name*
Identifies a security label that exists for the security policy that is associated with the table. The table must have a security policy associated with it (SQLSTATE 55064).

ADD *unique-constraint*
Defines a unique or primary key constraint. A primary key or unique constraint cannot be added to a table that is a subtable (SQLSTATE 429B3). If the table is a supertable at the top of the hierarchy, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*
Names the primary key or unique constraint. For more information, see *constraint-name* in "CREATE TABLE".

UNIQUE (*column-name***...,)**
Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL limits". No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on any of these types, or structured type can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). The set of columns in the unique key cannot be the same as the set of columns of the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine whether an existing index matches the unique key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. If a matching index definition is found, the description of the index is changed to indicate that it is required by the system and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected (the selection is arbitrary). If no matching index is found, a unique bidirectional index will automatically be created for the columns, as described in CREATE TABLE. See "Notes" on page 890 for details on index names associated with unique constraints.

PRIMARY KEY *...(column-name,)*
Defines a primary key composed of the identified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL limits". The table must not have a primary key and the identified columns must be defined as NOT NULL. No LOB, LONG VARCHAR, LONG VARGRAPHIC, DATALINK, distinct type based on any of these types, or structured type may be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.) Any existing values in the set of identified columns must be unique (SQLSTATE 23515).

A check is performed to determine if an existing index matches the primary key definition (ignoring any INCLUDE columns in the index). An index definition matches if it identifies the same set of columns without regard to the order of the columns or the direction (ASC/DESC) specifications. If a matching index definition is found, the description of the index is changed to indicate that it is the primary index, as required by the system, and it is changed to unique (after ensuring uniqueness) if it was a non-unique index. If the table has more than one matching index, an existing unique index is selected (the selection is arbitrary). If no matching index is found, a unique bidirectional index will automatically be created for the columns, as described in CREATE TABLE. See "Notes" on page 890 for details on index names associated with unique constraints.

Only one primary key can be defined on a table.

ADD *referential-constraint*
Defines a referential constraint. See *referential-constraint* in "CREATE TABLE".

ADD *check-constraint*
Defines a check constraint or functional dependency. See *check-constraint* in "CREATE TABLE".

ADD *distribution-clause*
Defines a distribution key. The table must be defined in a table space on a single-partition database partition group (SQLSTATE 55037) and must not already have a distribution key (SQLSTATE 42889). If a distribution key already exists for the table, the existing key must be dropped before adding the new distribution key. A distribution key cannot be added to a table that is a subtable (SQLSTATE 428DH) or to a table with a column of data type XML (SQLSTATE 42997).

DISTRIBUTE BY HASH (*column-name***...)**
Defines a distribution key using the specified columns. Each *column-name* must identify a column of the table, and the same column must not be identified more than once. The name cannot be qualified. A column cannot be used as part of a distribution key if the data type of the column is a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, DATALINK, XML, distinct type on any of these types, or structured type.

ADD RESTRICT ON DROP
Specifies that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

ADD MATERIALIZED QUERY

materialized-query-definition
Changes a regular table to a materialized query table for use during query optimization. The table specified by *table-name* must not:

- Be previously defined as a materialized query table
- Be a typed table
- Have any constraints, unique indexes, or triggers defined
- Reference a nickname that is marked with caching disabled
- Be referenced in the definition of another materialized query table
- Be referenced in the definition of a view that is enabled for query optimization

If *table-name* does not meet these criteria, an error is returned (SQLSTATE 428EW).

fullselect
Defines the query in which the table is based. The columns of the existing table must:

- have the same number of columns
- have exactly the same data types

- have the same column names in the same ordinal positions

as the result columns of *fullselect* (SQLSTATE 428EW). For details about specifying the *fullselect* for a materialized query table, see "CREATE TABLE". One additional restriction is that *table-name* cannot be directly or indirectly referenced in the fullselect.

refreshable-table-options
Specifies the refreshable options for altering a materialized query table.

DATA INITIALLY DEFERRED
The data in the table must be validated using the REFRESH TABLE or SET INTEGRITY statement.

REFRESH
Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified subselect is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The materialized query table can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY
Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool.

SYSTEM
Specifies that the data in the materialized query table is maintained by the system.

USER
Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL
Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE

statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

ADD PARTITION *add-partition*
Adds one or more data partitions to a partitioned table. If the specified table is not a partitioned table, an error is returned (SQLSTATE 428FT). The number of data partitions must not exceed 32 767.

partition-name
Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec
Specifies the range of values for the new data partition. This range must not overlap that of an existing data partition (SQLSTATE 56016). For a description of the starting-clause and the ending-clause, see "CREATE TABLE".

If the starting-clause is omitted, the new data partition is assumed to be at the end of the table. If the ending-clause is omitted, the new data partition is assumed to be at the start of the table. If the first column of the partitioning key is DESC, these assumptions are reversed.

IN *tablespace-name*
Specifies the table space where the data partition is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838). This can be a table space that is already being used for another data partition of the same table, or a table space that is currently not being used by this table. A table space must be specified when adding a data partition (SQLSTATE 42727).

LONG IN *tablespace-name*
Specifies the table space where the data partition containing long column data is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838).

ALTER FOREIGN KEY *constraint-name*
Alters the constraint attributes of the referential constraint *constraint-name*. The *constraint-name* must identify an existing referential constraint (SQLSTATE 42704).

ALTER CHECK *constraint-name*
Alters the constraint attributes of the check constraint or functional dependency *constraint-name*. The *constraint-name* must identify an existing check constraint or functional dependency (SQLSTATE 42704).

constraint-alteration
Options for changing attributes associated with referential or check constraints.

ENABLE QUERY OPTIMIZATION or **DISABLE QUERY OPTIMIZATION**
Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances.

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

ENFORCED or **NOT ENFORCED**
Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete.

ENFORCED
Change the constraint to ENFORCED. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621).

NOT ENFORCED
Change the constraint to NOT ENFORCED. This should only be specified if the table data is independently known to conform to the constraint.

ALTER *column-alteration*
Alters the definition of a column. Only the specified attributes will be altered; others will remain unchanged. Columns of a typed table cannot be altered (SQLSTATE 428DH).

column-name
Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the table (SQLSTATE 42703). The name must not be qualified. The name must not identify a column that is otherwise being added, altered, or dropped in the same ALTER TABLE statement (SQLSTATE 42711).

SET DATA TYPE *altered-data-type*
Specifies the new data type of the column. The new data type must be compatible with the existing data type of the column (SQLSTATE 42837). Table 135 lists the compatible data types. The "Reorg recommended" column identifies the data type alterations that will require table reorganization before a table can again be accessed (SQLSTATE 57016). In such cases, the column being altered cannot be part of a table containing an XML data type column (SQLSTATE 42997).

The data type of an identity column cannot be altered (SQLSTATE 42997).

The table cannot have data capture enabled (SQLSTATE 42997).

The specified length, precision, or scale can be greater than or equal to (but not less than) the existing length, precision, or scale (SQLSTATE 42837).

Table 135. Compatible Data Types

From type	To type	Valid for table partitioning key column	Valid for MDC organizing dimension column	Valid for partition key column	Reorg recommended
SMALLINT	INTEGER	yes	yes	yes	yes
SMALLINT	BIGINT	yes	yes	yes	yes
SMALLINT	DECIMAL (*p*, *m*); *p*-*m* > 4	yes	yes	no	yes

Table 135. Compatible Data Types (continued)

From type	To type	Valid for table partitioning key column	Valid for MDC organizing dimension column	Valid for partition key column	Reorg recommended
SMALLINT	REAL	yes	yes	no	yes
SMALLINT	DOUBLE	yes	yes	no	yes
INTEGER	BIGINT	yes	yes	yes	yes
INTEGER	DECIMAL (*p*, *m*); *p*-*m* > 9	yes	yes	no	yes
INTEGER	DOUBLE	yes	yes	no	yes
BIGINT	DECIMAL (*p*, *m*); *p*-*m* > 19	yes	yes	no	yes
REAL	DOUBLE	yes	yes	yes	yes
DECIMAL (*n*, *m*)	DECIMAL (*p*, *q*); *p* >= *n*; *q* >= *m*; (*p*-*q*) >= (*n*-*m*)	yes	yes	no	yes
CHARACTER (*n*)	CHARACTER (*n*+*x*)	no	yes	yes	yes
CHARACTER (*n*)	VARCHAR (*n*+*x*)	no	yes	yes	yes
VARCHAR (*n*)	CHARACTER (*n*+*x*)	no	yes	yes	yes
VARCHAR (*n*)	VARCHAR (*n*+*x*)	no	yes	yes	no
GRAPHIC (*n*)	GRAPHIC (*n*+*x*)	no	yes	yes	yes
GRAPHIC (*n*)	VARGRAPHIC (*n*+*x*)	no	yes	yes	yes
VARGRAPHIC (*n*)	VARGRAPHIC (*n*+*x*)	no	yes	yes	no
VARGRAPHIC (*n*)	GRAPHIC (*n*+*x*)	no	yes	yes	yes
BLOB (*n*)	BLOB (*n*+*x*)	n/a	n/a	n/a	yes
CLOB (*n*)	CLOB (*n*+*x*)	n/a	n/a	n/a	yes
DBCLOB (*n*)	DBCLOB (*n*+*x*)	n/a	n/a	n/a	yes

Altering a column must not make the total byte count of all columns exceed the maximum record size (SQLSTATE 54010). If the column is used in a unique constraint or an index, the new length must not cause the sum of the stored lengths for the unique constraint or index to exceed the index key length limit for the page size (SQLSTATE 54008). For column stored lengths, see "Byte Counts" in "CREATE TABLE". For key length limits, see "SQL limits".

Table 136. Cascaded Effects of Altering a Column

Operation	Effect
Altering a column that is referenced by a view or check constraint	The object is regenerated during alter processing. In the case of a view, function or method resolution for the object might be different after the alter operation, changing the semantics of the object. In the case of a check constraint, if the semantics of the object will change as a result of the alter operation, the operation fails.
Altering a column in a table that has a dependent package, trigger, or SQL routine	The object is marked invalid, and is revalidated on next use.
Altering the type of a column in a table that is referenced by an XSROBJECT enabled for decomposition	The XSROBJECT is marked inoperative for decomposition. Re-enabling the XSROBJECT might require readjustment of its mappings; following this, issue an ALTER XSROBJECT ENABLE DECOMPOSITION statement against the XSROBJECT.

SET *generated-column-spec*
Specifies the technique used to generate a value for the column. This can be in the form of a specific default value, an expression, or defining the column as an identity column. If an existing default for the column results from a different generation technique, that default must be dropped, which can be done in the same *column-alteration* using one of the DROP clauses.

default-clause
Specifies a new default value for the column that is to be altered. The column must not already be defined as the identity column or have a generation expression defined (SQLSTATE 42837). The specified default value must represent a value that could be assigned to the column in accordance with the rules for assignment as described in "Assignments and comparisons". Altering the default value does not change the value that is associated with this column for existing rows.

GENERATED ALWAYS or **GENERATED BY DEFAULT**
Specifies when the database manager is to generate values for the column. GENERATED BY DEFAULT specifies that a value is only to be generated when a value is not provided, or the DEFAULT keyword is used in an assignment to the column. GENERATED ALWAYS specifies that the database manager is to always generate a value for the column. GENERATED BY DEFAULT cannot be specified with a *generation-expression*.

identity-options
Specifies that the column is the identity column for the table. The column must not already be defined as the identity column, cannot have a generation expression, or cannot have an explicit default (SQLSTATE 42837). A table can only have a single identity column (SQLSTATE 428C1). The column must be specified as not nullable (SQLSTATE 42997), and the data type associated with the column must be an exact numeric data type with a scale of zero (SQLSTATE 42815). An exact numeric data type is one of: SMALLINT, INTEGER, BIGINT, DECIMAL, or NUMERIC with a

scale of zero, or a distinct type based on one of these types. For details on identity options, see "CREATE TABLE".

AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. The column must not already be defined with a generation expression, cannot be the identity column, or cannot have an explicit default (SQLSTATE 42837). The *generation-expression* must conform to the same rules that apply when defining a generated column. The result data type of the *generation-expression* must be assignable to the data type of the column (SQLSTATE 42821). The column must not be referenced in the distribution key column or in the ORGANIZE BY clause (SQLSTATE 42997).

SET EXPRESSION AS (*generation-expression***)**
Changes the expression for the column to the specified *generation-expression*. SET EXPRESSION AS requires the table to be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option. After the ALTER TABLE statement, the SET INTEGRITY statement with the IMMEDIATE CHECKED and FORCE GENERATED options must be used to update and check all the values in that column against the new expression. The column must already be defined as a generated column based on an expression (SQLSTATE 42837), and must not have appeared in the PARTITIONING KEY, DIMENSIONS, or KEY SEQUENCE clauses of the table (SQLSTATE 42997). The generation-expression must conform to the same rules that apply when defining a generated column. The result data type of the generation-expression must be assignable to the data type of the column (SQLSTATE 42821).

SET INLINE LENGTH *integer*
Changes the inline length of an existing structured type column. The inline length indicates the maximum byte size of an instance of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are handled.

The data type of *column-name* must be a structured type (SQLSTATE 42842).

The default inline length for a structured-type column is the inline length of its type (specified explicitly or by default in the CREATE TYPE statement). If the inline length of a structured type is less than 292, the value 292 is used for the inline length of the column.

The explicit inline length value can only be increased (SQLSTATE -1); must be at least 292; and cannot exceed 32672 (SQLSTATE 54010).

Altering the column must not make the total byte count of all columns exceed the maximum record size (SQLSTATE 54010).

Data that is already stored separately from the rest of the row will not be moved inline by this statement. To take advantage of the altered inline length of a structured type column, invoke the REORG command against the specified table after altering the inline length of its column.

SET NOT NULL
Specifies that the column cannot contain null values. No value for this column in existing rows of the table can be the null value (SQLSTATE

23502). This clause is not allowed if the column is specified in the foreign key of a referential constraint with a DELETE rule of SET NULL, and no other nullable columns exist in the foreign key (SQLSTATE 42831). Altering this attribute for a column requires table reorganization before further table access is allowed (SQLSTATE 57016). Note that because this operation requires validation of table data, it cannot be performed when the table is in reorg pending state (SQLSTATE 57016). The column being altered cannot be part of a table containing an XML data type column (SQLSTATE 42997). The table cannot have data capture enabled (SQLSTATE 42997).

SET GENERATED ALWAYS or **GENERATED BY DEFAULT**

Specifies when the database manager is to generate values for the column. GENERATED BY DEFAULT specifies that a value is only to be generated when a value is not provided or the DEFAULT keyword is used in an assignment to the column. GENERATED ALWAYS specifies that the database manager is to always generate a value for the column. The column must already be defined as a generated column based on an identity column; that is, defined with the AS IDENTITY clause (SQLSTATE 42837).

identity-alteration

Alters the identity attributes of the column. The column must be an identity column.

SET INCREMENT BY *numeric-constant*

Specifies the interval between consecutive values of the identity column. The next value to be generated for the identity column will be determined from the last assigned value with the increment applied. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence after the ALTER statement. If this value is 0 or positive, this is an ascending sequence after the ALTER statement.

SET NO MINVALUE or **MINVALUE** *numeric-constant*

Specifies the minimum value at which a descending identity column either cycles or stops generating values, or the value to which an ascending identity column cycles after reaching the maximum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MINVALUE

For an ascending sequence, the value is the original starting value. For a descending sequence, the value is the minimum value of the data type of the column.

MINVALUE *numeric-constant*

Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

SET NO MAXVALUE or **MAXVALUE** *numeric-constant*
Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or the value to which a descending identity column cycles after reaching the minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO MAXVALUE
For an ascending sequence, the value is the maximum value of the data type of the column. For a descending sequence, the value is the original starting value.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

SET NO CYCLE or **CYCLE**
Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CYCLE
Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached.

CYCLE
Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, then after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, duplicate values can be generated for an identity column. Although not required, if unique values are desired, a single-column unique index defined using the identity column will ensure uniqueness. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

SET NO CACHE or **CACHE** *integer-constant*
Specifies whether to keep some pre-allocated values in memory for faster access. This is a performance and tuning option. The column must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO CACHE
Specifies that values for the identity column are not to be pre-allocated. In a data sharing environment, if the identity values *must* be generated in order of request, the NO CACHE option must be used.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*

Specifies how many values of the identity sequence are pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value requires waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are lost (that is, they will never be used). The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of system failure.

The minimum value is 2 (SQLSTATE 42815).

SET NO ORDER or **ORDER**

Specifies whether the identity column values must be generated in order of request. The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837).

NO ORDER

Specifies that the identity column values do not need to be generated in order of request.

ORDER

Specifies that the identity column values must be generated in order of request.

RESTART or **RESTART WITH** *numeric-constant*

Resets the state of the sequence associated with the identity column. If WITH *numeric-constant* is not specified, the sequence for the identity column is restarted at the value that was specified, either implicitly or explicitly, as the starting value when the identity column was originally created.

The column must exist in the specified table (SQLSTATE 42703), and must already be defined with the IDENTITY attribute (SQLSTATE 42837). RESTART does *not* change the original START WITH value.

The *numeric-constant* is an exact numeric constant that can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA). The *numeric-constant* will be used as the next value for the column.

DROP IDENTITY

Drops the identity attributes of the column, making the column a simple numeric data type column. DROP IDENTITY is not allowed if the column is not an identity column (SQLSTATE 42837).

DROP EXPRESSION
Drops the generated expression attributes of the column, making the column a non-generated column. DROP EXPRESSION is not allowed if the column is not a generated expression column (SQLSTATE 42837).

DROP DEFAULT
Drops the current default for the column. The specified column must have a default value (SQLSTATE 42837).

DROP NOT NULL
Drops the NOT NULL attribute of the column, allowing the column to have the null value. This clause is not allowed if the column is specified in the primary key, or in a unique constraint of the table (SQLSTATE 42831). Altering this attribute for a column requires table reorganization before further table access is allowed (SQLSTATE 57016). The column being altered cannot be part of a table containing an XML data type column (SQLSTATE 42997). The table cannot have data capture enabled (SQLSTATE 42997).

ADD SCOPE
Add a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

typed-table-name
The name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

COMPRESS
Specifies whether or not default values for this column are to be stored more efficiently.

SYSTEM DEFAULT
Specifies that system default values (that is, the default values used for the data types when no specific values are specified) are to be stored using minimal space. If the table is not already set with the VALUE COMPRESSION attribute activated, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of the extra checking that is done.

Existing data in the column is not changed. Consider offline table reorganization to enable existing data to take advantage of storing system default values using minimal space.

OFF
Specifies that system default values are to be stored in the column as regular values. Existing data in the column is not changed. Offline reorganization is recommended to change existing data.

The base data type must not be DATE, TIME or TIMESTAMP (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

If the table being altered is a typed table, the column must not be inherited from a supertable (SQLSTATE 428DJ).

SECURED WITH *security-label-name*
Identifies a security label that exists for the security policy that is associated with the table. The table must have a security policy associated with it (SQLSTATE 55064).

DROP COLUMN SECURITY
Alters a column to make it a non-protected column.

ATTACH PARTITION *attach-partition*
Attaches another table as a new data partition. The data object of the table being attached becomes a new partition of the table being attached to. There is no data movement involved. The table is placed in set integrity pending state, and referential integrity checking is deferred until execution of a SET INTEGRITY statement.

partition-name
Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be ’PART’ followed by the character form of an integer value to make the name unique for the table.

boundary-spec
Specifies the range of values for the new data partition. This range must not overlap that of an existing data partition (SQLSTATE 56016). For a description of the starting-clause and the ending-clause, see “CREATE TABLE”.

If the starting-clause is omitted, the new data partition is assumed to be at the end of the table. If the ending-clause is omitted, the new data partition is assumed to be at the start of the table.

FROM *table-name1*
Specifies the table that is to be used as the source of data for the new partition. The table definition of *table-name1* cannot have multiple data partitions, and it must match the altered table in the following ways (SQLSTATE 428G3):

- The number of columns must be the same.
- The data types of the columns in the same ordinal position in the table must be the same.
- The nullability characteristic of the columns in the same ordinal position in the table must be the same.
- If the data is also distributed, it must be distributed over the same database partition group using the same distribution method.
- If the data in either table is organized, the organization must match.

After the data from *table-name1* is successfully attached, an operation equivalent to DROP TABLE *table-name1* is performed to remove this table, which no longer has data, from the database.

DETACH PARTITION *partition-name* **INTO** *table-name1*
Detaches the data partition *partition-name* from the altered table, and uses the

data partition to create a new table named *table-name1*. The data partition is logically attached to the new table without any data movement. The specified data partition cannot be the last remaining partition of the table being altered (SQLSTATE 428G2).

DROP PRIMARY KEY
Drops the definition of the primary key and all referential constraints dependent on this primary key. The table must have a primary key (SQLSTATE 42888).

DROP FOREIGN KEY *constraint-name*
Drops the referential constraint *constraint-name*. The *constraint-name* must identify a referential constraint (SQLSTATE 42704). For information on implications of dropping a referential constraint see "Notes" on page 890.

DROP UNIQUE *constraint-name*
Drops the definition of the unique constraint *constraint-name* and all referential constraints dependent on this unique constraint. The *constraint-name* must identify an existing UNIQUE constraint (SQLSTATE 42704). For information on implications of dropping a unique constraint, see "Notes" on page 890.

DROP CHECK *constraint-name*
Drops the check constraint *constraint-name*. The *constraint-name* must identify an existing check constraint defined on the table (SQLSTATE 42704).

DROP CONSTRAINT *constraint-name*
Drops the constraint *constraint-name*. The *constraint-name* must identify an existing check constraint, referential constraint, primary key, or unique constraint defined on the table (SQLSTATE 42704). For information on implications of dropping a constraint, see "Notes" on page 890.

DROP COLUMN
Drops the identified column from the table. The table must not be a typed table (SQLSTATE 428DH). The table cannot have data capture enabled (SQLSTATE 42997). Dropping a column requires table reorganization before further table access is allowed.

column-name
Identifies the column that is to be dropped. The column name must not be qualified. The name must identify a column of the specified table (SQLSTATE 42703). The name must not identify the only column of the table (SQLSTATE 42814). The name must not identify a column that is part of the table's distribution key, table partitioning key, or organizing dimensions (SQLSTATE 42997). The name must not identify a column that is part of a table containing an XML data type column (SQLSTATE 42997).

CASCADE
Specifies that any views, indexes, triggers, SQL functions, or constraints that are dependent on the column being dropped are also dropped, or that any decomposition-enabled XSROBJECTs that are dependent on the table containing the column are made inoperative for decomposition. A trigger is dependent on the column if it is referenced in the UPDATE OF column list, or anywhere in the triggered action. A decomposition-enabled XSROBJECT is dependent on a table if it contains a mapping of an XML element or attribute to the table. If an SQL function is dependent on another database object, it might not be possible to drop the function by means of the CASCADE option. CASCADE is the default.

RESTRICT
Specifies that the column cannot be dropped if any views, indexes,

triggers, or constraints are dependent on the column, or if any decomposition-enabled XSROBJECT is dependent on the table that contains the column (SQLSTATE 42893). A trigger is dependent on the column if it is referenced in the UPDATE OF column list, or anywhere in the triggered action. A decomposition-enabled XSROBJECT is dependent on a table if it contains a mapping of an XML element or attribute to the table. The first dependent object that is detected is identified in the administration log.

Table 137. Cascaded Effects of Dropping a Column

Operation	RESTRICT Effect	CASCADE Effect
Dropping a column that is referenced by a view or a trigger	Dropping the column is not allowed.	The object and all objects that are dependent on that object are dropped.
Dropping a column that is referenced in the key of an index	If all columns that are referenced in the index are dropped in the same ALTER TABLE statement, dropping the index is allowed. Otherwise, dropping the column is not allowed.	The index is dropped.
Dropping a column that is referenced in a unique constraint	If all columns that are referenced in the unique constraint are dropped in the same ALTER TABLE statement, and the unique constraint is not referenced by a referential constraint, the columns and the constraint are dropped. (The index that is used to satisfy the constraint is also dropped.) Otherwise, dropping the column is not allowed.	The unique constraint and any referential constraints that reference that unique constraint are dropped. (Any indexes that are used by those constraints are also dropped).
Dropping a column that is referenced in a referential constraint	If all columns that are referenced in the referential constraint are dropped in the same ALTER TABLE statement, the columns and the constraint are dropped. Otherwise, dropping the column is not allowed.	The referential constraint is dropped.
Dropping a column that is referenced by a system-generated column that is not being dropped.	Dropping the column is not allowed.	Dropping the column is not allowed.
Dropping a column that is referenced in a check constraint	Dropping the column is not allowed.	The check constraint is dropped.

Table 137. Cascaded Effects of Dropping a Column (continued)

Operation	RESTRICT Effect	CASCADE Effect
Dropping a column that is referenced in a decomposition-enabled XSROBJECT	Dropping the column is not allowed.	The XSROBJECT is marked inoperative for decomposition. Re-enabling the XSROBJECT might require readjustment of its mappings; following this, issue an ALTER XSROBJECT ENABLE DECOMPOSITION statement against the XSROBJECT.

DROP PARTITIONING KEY
Drops the partitioning key. The table must have a partitioning key and must be in a table space defined on a single-partition database partition group (SQLSTATE 42888).

DROP RESTRICT ON DROP
Removes the restriction, if there is one, on dropping the table and the table space that contains the table.

DROP DISTRIBUTION
Drops the distribution definition for the table. The table must have a distribution definition (SQLSTATE 428FT). The table space for the table must be defined on a single partition database partition group.

DROP MATERIALIZED QUERY
Changes a materialized query table so that it is no longer considered to be a materialized query table. The table specified by *table-name* must be defined as a materialized query table that is not replicated (SQLSTATE 428EW). The definition of the columns of *table-name* is not changed, but the table can no longer be used for query optimization, and the REFRESH TABLE statement can no longer be used.

DATA CAPTURE
Indicates whether extra information for data replication is to be written to the log.

If the table is a typed table, then this option is not supported (SQLSTATE 428DH for root tables or 428DR for other subtables).

Data capture is incompatible with row compression (SQLSTATE 42997).

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the table is defined to allow data on a database partition other than the catalog partition (multiple partition database partition group, or database partition group with database partitions other than the catalog partition), then this option is not supported (SQLSTATE 42997).

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

INCLUDE LONGVAR COLUMNS
Allows data replication utilities to capture changes made to LONG VARCHAR or LONG VARGRAPHIC columns. The clause may be specified for tables that do not have any LONG VARCHAR or LONG VARGRAPHIC columns since it is possible to ALTER the table to include such columns.

ACTIVATE NOT LOGGED INITIALLY
Activates the NOT LOGGED INITIALLY attribute of the table for this current unit of work.

Any changes made to the table by an INSERT, DELETE, UPDATE, CREATE INDEX, DROP INDEX, or ALTER TABLE in the same unit of work after the table is altered by this statement are not logged. Any changes made to the system catalog by the ALTER statement in which the NOT LOGGED INITIALLY attribute is activated are logged. Any subsequent changes made in the same unit of work to the system catalog information are logged.

At the completion of the current unit of work, the NOT LOGGED INITIALLY attribute is deactivated and all operations that are done on the table in subsequent units of work are logged.

If using this feature to avoid locks on the catalog tables while inserting data, it is important that only this clause be specified on the ALTER TABLE statement. Use of any other clause in the ALTER TABLE statement will result in catalog locks. If no other clauses are specified for the ALTER TABLE statement, then only a SHARE lock will be acquired on the system catalog tables. This can greatly reduce the possibility of concurrency conflicts for the duration of time between when this statement is executed and when the unit of work in which it was executed is ended.

If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

For more information about the NOT LOGGED INITIALLY attribute, see the description of this attribute in “CREATE TABLE”.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred and can only be dropped. Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

WITH EMPTY TABLE
Causes all data currently in table to be removed. Once the data has been removed, it cannot be recovered except through use of the RESTORE facility. If the unit of work in which this alter statement was issued is rolled back, the table data will not be returned to its original state.

When this action is requested, no DELETE triggers defined on the affected table are fired. Any indexes that exist on the table are also deleted.

A partitioned table with attached data partitions cannot be emptied (SQLSTATE 42928).

PCTFREE *integer*
Specifies the percentage of each page that is to be left as free space during a

load or a table reorganization operation. The first row on each page is added without restriction. When additional rows are added to a page, at least *integer* percent of the page is left as free space. The PCTFREE value is considered only by the load and table reorg utilities. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the system catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

LOCKSIZE
Indicates the size (granularity) of locks used when the table is accessed. Use of this option in the table definition will not prevent normal lock escalation from occurring. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ROW
Indicates the use of row locks. This is the default lock size when a table is created.

BLOCKINSERT
Indicates the use of block locks during insert operations. This means that the appropriate exclusive lock is acquired on the block before insertion, and row locking is not done on the inserted row. This option is useful when separate transactions are inserting into separate cells in the table. Transactions inserting into the same cells can still do so concurrently, but will insert into distinct blocks, and this can impact the size of the cell if more blocks are needed. This option is only valid for MDC tables (SQLSTATE 628N).

TABLE
Indicates the use of table locks. This means that the appropriate share or exclusive lock is acquired on the table, and that intent locks (except intent none) are not used. For partitioned tables, this lock strategy is applied to both the table lock and the data partition locks for any data partitions that are accessed. Use of this value can improve the performance of queries by limiting the number of locks that need to be acquired. However, concurrency is also reduced, because all locks are held over the complete table.

APPEND
Indicates whether data is appended to the end of the table data or placed where free space is available in data pages. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ON
Indicates that table data will be appended and information about free space on pages will not be kept. The table must not have a clustered index (SQLSTATE 428CA).

OFF
Indicates that table data will be placed where there is available space. This is the default when a table is created.

The table should be reorganized after setting APPEND OFF since the information about available free space is not accurate and may result in poor performance during insert.

VOLATILE CARDINALITY or **NOT VOLATILE CARDINALITY**
Indicates to the optimizer whether or not the cardinality of table *table-name* can

vary significantly at run time. Volatility applies to the number of rows in the table, not to the table itself. CARDINALITY is an optional keyword. The default is NOT VOLATILE.

VOLATILE
Specifies that the cardinality of table *table-name* can vary significantly at run time, from empty to large. To access the table, the optimizer will use an index scan (rather than a table scan, regardless of the statistics) if that index is index-only (all referenced columns are in the index), or that index is able to apply a predicate in the index scan. The list prefetch access method will not be used to access the table. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

NOT VOLATILE
Specifies that the cardinality of *table-name* is not volatile. Access plans to this table will continue to be based on existing statistics and on the current optimization level.

COMPRESS
Specifies whether or not data compression applies to the rows of the table.

YES
Specifies that data row compression is to be enabled. For tables with no pre-existing compression dictionary, the data rows in the table are not subject to compression until a compression dictionary has been created with the non-inplace reorg utility or the inspect rowcompestimate utility. For tables that already have a compression dictionary, compression is reactivated to use this dictionary, and data row compression applies to any subsequent data movement operation.

NO
Specifies that data row compression is to be disabled. Insert and update operations against the table will no longer be subject to compression. Any rows in the table that are in compressed format remain in compressed format until they are converted to non-compressed format when they are updated. A non-inplace reorganization of the table decompresses all rows that are compressed. If a compression dictionary exists, it is discarded during table reinitialization or truncation (such as, for example, a replace operation).

VALUE COMPRESSION
This determines the row format that is to be used. Each data type has a different byte count depending on the row format that is used. For more information, see “Byte Counts” in “CREATE TABLE”. An update operation causes an existing row to be changed to the new row format. Offline table reorganization is recommended to improve the performance of update operations on existing rows. This can also result in the table taking up less space. If the row size, calculated using the appropriate column in the table named “Byte Counts of Columns by Data Type” (see “CREATE TABLE”), would no longer fit within the row size limit, as indicated in the table named “Limits for Number of Columns and Row Size In Each Table Space Page Size”, an error is returned (SQLSTATE 54010). If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

ACTIVATE
The NULL value is stored using three bytes. This is the same or less space than when VALUE COMPRESSION is not active for columns of all data

types, with the exception of CHAR(1). Whether or not a column is defined as nullable has no affect on the row size calculation. The zero-length data values for columns whose data type is VARCHAR, VARGRAPHIC, LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, or BLOB are to be stored using two bytes only, which is less than the storage required when VALUE COMPRESSION is not active. When a column is defined using the COMPRESS SYSTEM DEFAULT option, this also allows the system default value for the column to be stored using three bytes of total storage. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation when updating to or from NULL, a zero-length value, or the system default value.

DEACTIVATE
The NULL value is stored with space set aside for possible future updates. This space is not set aside for varying-length columns. It also does not support efficient storage of system default values for a column. If columns already exist with the COMPRESS SYSTEM DEFAULT attribute, a warning is returned (SQLSTATE 01648).

LOG INDEX BUILD
Specifies the level of logging that is to be performed during create, recreate, or reorganize index operations on this table.

NULL
Specifies that the value of the *logindexbuild* database configuration parameter will be used to determine whether or not index build operations are to be completely logged. This is the default when the table is created.

OFF
Specifies that any index build operations on this table will be logged minimally. This value overrides the setting of the *logindexbuild* database configuration parameter.

ON
Specifies that any index build operations on this table will be logged completely. This value overrides the setting of the *logindexbuild* database configuration parameter.

Rules:

- Any unique or primary key constraint defined on the table must be a superset of the distribution key, if there is one (SQLSTATE 42997).
- Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).
- A column can only be referenced in one ADD, ALTER, or DROP COLUMN clause in a single ALTER TABLE statement (SQLSTATE 42711).
- A column length or data type cannot be altered, nor can the column be dropped, if the table has any materialized query tables that are dependent on the table (SQLSTATE 42997).
- VARCHAR and VARGRAPHIC columns that have been altered to be greater than 4000 and 2000, respectively, must not be used as input parameters in functions in the SYSFUN schema (SQLSTATE 22001).
- A column length cannot be altered if the table has any views enabled for query optimization that are dependent on the table (SQLSTATE 42997).
- The table must be put in set integrity pending state, using the SET INTEGRITY statement with the OFF option (SQLSTATE 55019), before:
  - Adding a column with a generation expression
  - Altering the generated expression of a column

  - Changing a column to have a generated expression
- A column of data type XML cannot be added to a table if there are type 1 indexes on that table (SQLSTATE 42997). The indexes can be converted to type 2 indexes using the REORG INDEXES command with the CONVERT option.
- An existing column cannot be altered to become of type DB2SECURITYLABEL (SQLSTATE 42837).
- Defining a column of type DB2SECURITYLABEL fails if the table does not have a security policy associated with it (SQLSTATE 55064).
- A column of type DB2SECURITYLABEL cannot be altered or dropped (SQLSTATE 42817).
- An ALTER TABLE operation to mark a table as protected fails if there exists an MQT that depends on that table (SQLSTATE 55067).
- Attaching a partition to a protected partitioned table fails if the source table and the target table are not protected using the same security policy, have the same row security label column, and have the same set of protected columns (SQLSTATE 428GE).
- If a generated column is referenced in a table partitioning key, the generated column expression cannot be altered (SQLSTATE 42837).

Notes:

- A *REORG-recommended operation* has occured when changes resulting from an ALTER TABLE statement affect the row format of the data. When this occurs, most subsequent operations on the table are restricted until a table reorganization operation completes successfully. Up to three ALTER TABLE statements of this type can execute against a table before reorganization must be done (SQLSTATE 57016). Multiple alterations that would constitute a REORG-recommended operation can be made as part of a single ALTER TABLE statement (one per column); this is considered to be a single REORG-recommended operation. For example, dropping two columns in a single ALTER TABLE statement is not considered to be two REORG-recommended operations. Dropping two columns in two separate ALTER TABLE statements, however, would be regarded as two statements that contain REORG-recommended operations.
- The following table operations are allowed after a successful REORG-recommended operation has occurred:
  - ALTER TABLE, where no row data validation is required. However, the following operations are not allowed (SQLSTATE 57007):
    - ADD CHECK CONSTRAINT
    - ADD REFERENTIAL CONSTRAINT
    - ADD UNIQUE CONSTRAINT
    - ALTER COLUMN SET NOT NULL
  - DROP TABLE
  - RENAME TABLE
  - REORG TABLE
  - TRUNCATE TABLE
  - Table scan access of table data
- Altering a table to make it a materialized query table will put the table in set integrity pending state. If the table is defined as REFRESH IMMEDIATE, the table must be taken out of set integrity pending state before INSERT, DELETE, or UPDATE commands can be invoked on the table referenced by the fullselect. The table can be taken out of set integrity pending state by using REFRESH

TABLE or SET INTEGRITY, with the IMMEDIATE CHECKED option, to completely refresh the data in the table based on the fullselect. If the data in the table accurately reflects the result of the fullselect, the IMMEDIATE UNCHECKED option of SET INTEGRITY can be used to take the table out of set integrity pending state.

- Altering a table to change it to a REFRESH IMMEDIATE materialized query table will cause any packages with INSERT, DELETE, or UPDATE usage on the table referenced by the fullselect to be invalidated.
- Altering a table to change from a materialized query table to a regular table will cause any packages dependent on the table to be invalidated.
- Altering a table to change from a MAINTAINED BY FEDERATED_TOOL materialized query table to a regular table will not cause any change in the subscription setup of the replication tool. Because a subsequent change to a MAINTAINED BY SYSTEM materialized query table will cause the replication tool to fail, you must change the subscription setting when changing a MAINTAINED BY FEDERATED_TOOL materialized query table.
- If a deferred materialized query table is associated with a staging table, the staging table will be dropped if the materialized query table is altered to a regular table.
- ADD column clauses are processed prior to all other clauses. Other clauses are processed in the order that they are specified.
- Any columns added through an alter table operation will not automatically be added to any existing view of the table.
- Adding or attaching a data partition to a partitioned table, or detaching a data partition from a partitioned table causes any packages that are dependent on that table to be invalidated.
- To drop the partitioning for a table, the table must be dropped and then recreated.
- To drop the organization for a table, the table must be dropped and then recreated.
- When an index is automatically created for a unique or primary key constraint, the database manager will try to use the specified constraint name as the index name with a schema name that matches the schema name of the table. If this matches an existing index name or no name for the constraint was specified, the index is created in the SYSIBM schema with a system-generated name formed of ″SQL″ followed by a sequence of 15 numeric characters generated by a timestamp based function.
- When an index is created on a partitioned table with attached data partitions, the index will not include the data in the attached data partitions. Use the SET INTEGRITY statement to maintain all indexes for all attached data partitions.
- Any table that may be involved in a DELETE operation on table T is said to be *delete-connected* to T. Thus, a table is delete-connected to T if it is a dependent of T or it is a dependent of a table in which deletes from T cascade.
- A package has an insert (update/delete) usage on table T if records are inserted into (updated in/deleted from) T either directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements. Similarly, a package has an update usage on a column if the column is modified directly by a statement in the package, or indirectly through constraints or triggers executed by the package on behalf of one of its statements.

- In a federated system, a remote base table that was created using transparent DDL can be altered. However, transparent DDL does impose some limitations on the modifications that can be made:
  - A remote base table can only be altered by adding new columns or specifying a primary key.
  - Specific clauses supported by transparent DDL include:
    - ADD COLUMN *column-definition*
    - NOT NULL and PRIMARY KEY in the *column-options* clause
    - ADD *unique-constraint* (PRIMARY KEY only)
  - You cannot specify a comment on an existing column in a remote base table.
  - An existing primary key in a remote base table cannot be altered or dropped.
  - Altering a remote base table invalidates any packages that are dependent on the nickname associated with that remote base table.
  - The remote data source must support the changes being requested through the ALTER TABLE statement. Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.
  - An attempt to alter a remote base table that was not created using transparent DDL returns an error.
- Any changes, whether implicit or explicit, to primary key, unique keys, or foreign keys might have the following effects on packages, indexes, and other foreign keys.
  - If a primary key or unique key is added:
    - There is no effect on packages, foreign keys, or existing unique keys. (If the primary or unique key uses an existing unique index that was created in a previous version and has not been converted to support deferred uniqueness, the index is converted, and packages with update usage on the associated table are invalidated.)
  - If a primary key or unique key is dropped:
    - The index is dropped if it was automatically created for the constraint. Any packages dependent on the index are invalidated.
    - The index is set back to non-unique if it was converted to unique for the constraint and it is no longer system-required. Any packages dependent on the index are invalidated.
    - The index is set to no longer system required if it was an existing unique index used for the constraint. There is no effect on packages.
    - All dependent foreign keys are dropped. Further action is taken for each dependent foreign key, as specified in the next item.
  - If a foreign key is added, dropped, or altered from NOT ENFORCED to ENFORCED (or ENFORCED to NOT ENFORCED):
    - All packages with an insert usage on the object table are invalidated.
    - All packages with an update usage on at least one column in the foreign key are invalidated.
    - All packages with a delete usage on the parent table are invalidated.
    - All packages with an update usage on at least one column in the parent key are invalidated.
  - If a foreign key or a functional dependency is altered from ENABLE QUERY OPTIMIZATION to DISABLE QUERY OPTIMIZATION:
    - All packages with dependencies on the constraint for optimization purposes are invalidated.

- Adding a column to a table will result in invalidation of all packages with insert usage on the altered table. If the added column is the first user-defined structured type column in the table, packages with DELETE usage on the altered table will also be invalidated.
- Adding a check or referential constraint to a table that already exists and that is not in set integrity pending state, or altering the existing check or referential constraint from NOT ENFORCED to ENFORCED on an existing table that is not in set integrity pending state will cause the existing rows in the table to be immediately evaluated against the constraint. If the verification fails, an error is returned (SQLSTATE 23512). If a table is in set integrity pending state, adding a check or referential constraint, or altering a constraint from NOT ENFORCED to ENFORCED will not immediately lead to the enforcement of the constraint. Issue the SET INTEGRITY statement with the IMMEDIATE CHECKED option to begin enforcing the constraint.
- Adding, altering, or dropping a check constraint will result in invalidation of all packages with either an insert usage on the object table, an update usage on at least one of the columns involved in the constraint, or a select usage exploiting the constraint to improve performance.
- Adding a distribution key invalidates all packages with an update usage on at least one of the columns of the distribution key.
- A distribution key that was defined by default as the first column of the primary key is not affected by dropping the primary key and adding a different primary key.
- Dropping a column or changing its data type removes all runstats information from the table being altered. Runstats should be performed on the table after it is again accessible. The statistical profile of the table is preserved if the table does not contain a column that was explicitly dropped.
- Altering a column (to increase its length or change its data type or nullability attribute) or dropping a column invalidates all packages that reference (directly or indirectly through a referential constraint or trigger) its table.
- Altering a column (to increase its length or change its data type or nullability attribute) regenerates views (except typed views) that are dependent on its table. If a problem occurs while regenerating such a view, an error is returned (SQLSTATE 56098). Any typed views that are dependent on the table are marked inoperative.
- Altering a column to increase its length or change its data type marks all dependent triggers and SQL functions as invalid; they are implicitly recompiled on next use. If a problem occurs while regenerating such an object, an error is returned (SQLSTATE 56098).
- Altering a column (to increase its length or change its data type or nullability attribute) might cause errors (SQLSTATE 54010) while processing a trigger or an SQL function when a statement involving the trigger or SQL function is prepared or bound. This can occur if the row length based on the sum of the lengths of the transition variables and transition table columns is too long. If such a trigger or SQL function is dropped, a subsequent attempt to recreate it returns an error (SQLSTATE 54040).
- Altering a structured type column to increase the inline length will invalidate all packages that reference the table, either directly or indirectly through a referential constraint or trigger.
- Altering a structured type column to increase the inline length will regenerate views that are dependent on the table.
- Changing the LOCKSIZE for a table will result in invalidation of all packages that have a dependency on the altered table.

- The ACTIVATE NOT LOGGED INITIALLY clause cannot be used when DATALINK columns with the FILE LINK CONTROL attribute are being added to the table (SQLSTATE 42613).
- Changing VOLATILE or NOT VOLATILE CARDINALITY will result in invalidation of all packages that have a dependency on the altered table.
- ***Replication:*** Exercise caution when increasing the length or changing the data type of a column. The change data table that is associated with an application table might already be at or near the DB2 row size limit. The change data table should be altered before the application table, or the two tables should be altered within the same unit of work, to ensure that the alteration can be completed for both tables. Consideration should be given to copies, which might also be at or near the row size limit, or reside on platforms which lack the ability to increase the length of an existing column.

  If the change data table is not altered before the Capture program processes log records with the altered attributes, the Capture program will likely fail. If a copy containing the altered column is not altered before the subscription maintaining the copy runs, the subscription will likely fail.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - The ADD keyword is optional for:
      - Unnamed PRIMARY KEY constraints
      - Unnamed referential constraints
      - Referential constraints whose name follows the phrase FOREIGN KEY
    - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
    - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
    - SET SUMMARY AS can be specified in place of SET MATERIALIZED QUERY AS
    - SET MATERIALIZED QUERY AS DEFINITION ONLY can be specified in place of DROP MATERIALIZED QUERY
    - SET MATERIALIZED QUERY AS (fullselect) can be specified in place of ADD MATERIALIZED QUERY (fullselect)
    - ADD PARTITIONING KEY can be specified in place of ADD DISTRIBUTE BY HASH; the optional USING HASHING clause can also still be specified in this case
  - For compatibility with previous versions of DB2 and for consistency:
    - A comma can be used to separate multiple options in the *identity-alteration* clause
  - For compatibility with DB2 UDB for z/OS:
    - PART can be specified in place of PARTITION
    - VALUES can be specified in place of ENDING AT
  - The following syntax is also supported:
    - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER
- When detaching a partition from a protected table, the target table automatically created by DB2 will be protected in exactly the same way the source table is protected.

- When a table is altered such that it becomes protected with row level granularity, any cached dynamic SQL sections that depend on such a table are invalidated. Similarly, any packages that depend on such a table are also invalidated.
- When a column of a table, T, is altered such that it becomes a protected column, any cached dynamic SQL sections that depend on table T are invalidated. Similarly, any packages that depend on table T are also invalidated.
- When a column of a table, T, is altered such that it becomes a non protected column, any cached dynamic SQL sections that depend on table T are invalidated. Similarly, any packages that depend on table T are also invalidated.
- For existing rows in the table, the value of the security label column defaults to the security label for write access of the session authorization ID at the time the ALTER statement that adds a row security label column is executed.

Examples:

Example 1: Add a new column named RATING, which is one character long, to the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
   ADD RATING CHAR(1)
```

Example 2: Add a new column named SITE_NOTES to the PROJECT table. Create SITE_NOTES as a varying-length column with a maximum length of 1000 bytes. The values of the column do not have an associated character set and therefore should not be converted.

```
ALTER TABLE PROJECT
   ADD SITE_NOTES  VARCHAR(1000) FOR BIT DATA
```

Example 3: Assume a table called EQUIPMENT exists defined with the following columns:

```
Column Name        Data Type
EQUIP_NO           INT
EQUIP_DESC         VARCHAR(50)
LOCATION           VARCHAR(50)
EQUIP_OWNER        CHAR(3)
```

Add a referential constraint to the EQUIPMENT table so that the owner (EQUIP_OWNER) must be a department number (DEPTNO) that is present in the DEPARTMENT table. DEPTNO is the primary key of the DEPARTMENT table. If a department is removed from the DEPARTMENT table, the owner (EQUIP_OWNER) values for all equipment owned by that department should become unassigned (or set to null). Give the constraint the name DEPTQUIP.

```
ALTER TABLE EQUIPMENT
     ADD CONSTRAINT DEPTQUIP
      FOREIGN KEY (EQUIP_OWNER)
        REFERENCES DEPARTMENT
           ON DELETE SET NULL
```

Also, an additional column is needed to allow the recording of the quantity associated with this equipment record. Unless otherwise specified, the EQUIP_QTY column should have a value of 1 and must never be null.

```
ALTER TABLE EQUIPMENT
  ADD COLUMN EQUIP_QTY
  SMALLINT NOT NULL DEFAULT 1
```

Example 4: Alter table EMPLOYEE. Add the check constraint named REVENUE defined so that each employee must make a total of salary and commission greater than $30,000.

```
ALTER TABLE EMPLOYEE
  ADD CONSTRAINT REVENUE
  CHECK (SALARY + COMM > 30000)
```

Example 5: Alter table EMPLOYEE. Drop the constraint REVENUE which was previously defined.

```
ALTER TABLE EMPLOYEE
  DROP CONSTRAINT REVENUE
```

Example 6: Alter a table to log SQL changes in the default format.

```
ALTER TABLE SALARY1
  DATA CAPTURE NONE
```

Example 7: Alter a table to log SQL changes in an expanded format.

```
ALTER TABLE SALARY2
  DATA CAPTURE CHANGES
```

Example 8: Alter the EMPLOYEE table to add 4 new columns with default values.

```
ALTER TABLE EMPLOYEE
   ADD COLUMN HEIGHT MEASURE   DEFAULT MEASURE(1)
   ADD COLUMN BIRTHDAY BIRTHDATE DEFAULT DATE('01-01-1850')
   ADD COLUMN FLAGS BLOB(1M)  DEFAULT BLOB(X'01')
   ADD COLUMN PHOTO PICTURE   DEFAULT BLOB(X'00')
```

The default values use various function names when specifying the default. Since MEASURE is a distinct type based on INTEGER, the MEASURE function is used. The HEIGHT column default could have been specified without the function since the source type of MEASURE is not BLOB or a datetime data type. Since BIRTHDATE is a distinct type based on DATE, the DATE function is used (BIRTHDATE cannot be used here). For the FLAGS and PHOTO columns the default is specified using the BLOB function even though PHOTO is a distinct type. To specify a default for BIRTHDAY, FLAGS and PHOTO columns, a function must be used because the type is a BLOB or a distinct type sourced on a BLOB or datetime data type.

Example 9: A table called CUSTOMERS is defined with the following columns:

Column Name	Data Type
BRANCH_NO	SMALLINT
CUSTOMER_NO	DECIMAL(7)
CUSTOMER_NAME	VARCHAR(50)

In this table, the primary key is made up of the BRANCH_NO and CUSTOMER_NO columns. To distribute the table, you will need to create a distribution key for the table. The table must be defined in a table space on a single-node database partition group. The primary key must be a superset of the distribution key columns: at least one of the columns of the primary key must be used as the distribution key. Make BRANCH_NO the distribution key as follows:

```
ALTER TABLE CUSTOMERS
  ADD DISTRIBUTE BY HASH (BRANCH_NO)
```

Example 10: A remote table EMPLOYEE was created in a federated system using transparent DDL. Alter the remote table EMPLOYEE to add the columns PHONE_NO and WORK_DEPT; also add a primary key on the existing column EMP_NO and the new column WORK_DEPT.

```
ALTER TABLE EMPLOYEE
  ADD COLUMN PHONE_NO CHAR(4) NOT NULL
  ADD COLUMN WORK_DEPT CHAR(3)
  ADD PRIMARY KEY (EMP_NO, WORK_DEPT)
```

Example 11: Alter the DEPARTMENT table to add a functional dependency FD1, then drop the functional dependency FD1 from the DEPARTMENT table.

```
ALTER TABLE DEPARTMENT
  ADD CONSTRAINT FD1
    CHECK ( DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED

ALTER TABLE DEPARTMENT
  DROP CHECK FD1
```

Example 12: Change the default value for the WORKDEPT column in the EMPLOYEE table to 123.

```
ALTER TABLE EMPLOYEE
  ALTER COLUMN WORKDEPT
    SET DEFAULT '123'
```

Example 13: Alter the table EMPLOYEE to make it a protected table.

```
ALTER TABLE EMPLOYEE
  ADD SECURITY POLICY DATA_ACCESS
```

Example 14: Alter the table EMPLOYEE to protect the SALARY column.

```
ALTER TABLE EMPLOYEE
  ALTER COLUMN SALARY
  SECURED WITH EMPLOYEESECLABEL
```

Example 15: Assume that you have a table named SALARY_DATA that is defined with the following columns:

```
Column Name          Data Type
-----------          ---------
EMP_NAME             VARCHAR(50) NOT NULL
EMP_ID               SMALLINT NOT NULL
EMP_POSITION         VARCHAR(100) NOT NULL
SALARY               DECIMAL(5,2)
PROMOTION_DATE       DATE NOT NULL
```

Change this table to allow salaries to be stored in a DECIMAL(6,2) column, make PROMOTION_DATE an optional field that can be set to the null value, and remove the EMP_POSITION column.

```
ALTER TABLE SALARY_DATA
  ALTER COLUMN SALARY SET DATA TYPE DECIMAL(6,2)
  ALTER COLUMN PROMOTION_DATE DROP NOT NULL
  DROP COLUMN EMP_POSITION
```

Related concepts:

- "Database authorities" on page 74

Related tasks:

- "Adding data partitions to partitioned tables" in *Administration Guide: Implementation*
- "Altering a table" in *Administration Guide: Implementation*
- "Attaching a data partition" in *Administration Guide: Implementation*
- "Dropping a data partition" in *Administration Guide: Implementation*
- "Detaching a data partition" in *Administration Guide: Implementation*

- "Rotating data in a partitioned table" in *Administration Guide: Implementation*
- "Recovering a dropped table" in *Data Recovery and High Availability Guide and Reference*

Related reference:
- "Examples of rolling in and rolling out partitioned table data" in *Administration Guide: Implementation*
- "ALTER TYPE (Structured) statement" in *SQL Reference, Volume 2*
- "CREATE TABLE " on page 956
- "Assignments and comparisons" in *SQL Reference, Volume 1*
- "SQL and XQuery limits" in *SQL Reference, Volume 1*
- "ALTOBJ procedure" in *Administrative SQL Routines and Views*

Related samples:
- "dbrecov.sqc -- How to recover a database (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "dbrecov.sqC -- How to recover a database (C++)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "TbGenCol.java -- How to use generated columns (JDBC)"

ALTER TABLESPACE

The ALTER TABLESPACE statement is used to modify an existing table space in the following ways:
- Add a container to, or drop a container from a DMS table space; that is, a table space created with the MANAGED BY DATABASE option.
- Modify the size of a container in a DMS table space.
- Add a container to an SMS table space on a database partition that currently has no containers.
- Modify the PREFETCHSIZE setting for a table space.
- Modify the BUFFERPOOL used for tables in the table space.
- Modify the OVERHEAD setting for a table space.
- Modify the TRANSFERRATE setting for a table space.
- Modify the file system caching policy for a table space.

Invocation:

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax:

```
>>--ALTER TABLESPACE--tablespace-name---------------------------------------------------------->

    .-------------------------------------------------------------------------------------.
    V                                                                                     |
>----+-ADD--+--------------------------+--| database-container-clause |--+-----------------------------+-+-+-----><
     |      |                          |                                 '-| on-db-partitions-clause |-' | |
     |      '-TO STRIPE SET--stripeset-'                                                                 | |
     |      '-| system-container-clause |--| on-db-partitions-clause |-----------------------------------' |
     +-BEGIN NEW STRIPE SET--| database-container-clause |--+-----------------------------+---------------+
     |                                                       '-| on-db-partitions-clause |-'              |
     +-DROP--| drop-container-clause |--+-----------------------------+--------------------------------+
     |                                  '-| on-db-partitions-clause |-'                                |
     +-+-EXTEND-+--+-| database-container-clause |-+--+-----------------------------+------------------+
     | +-REDUCE-+  '-| all-containers-clause |-----'  '-| on-db-partitions-clause |-'                  |
     | '-RESIZE-'                                                                                      |
     +-PREFETCHSIZE--+-AUTOMATIC---------+-----------------------------------------------------------+
     |               +-number-of-pages---+                                                           |
     |               '-integer--+-K-+----'                                                           |
     |                          +-M-+                                                                |
     |                          '-G-'                                                                |
     +-BUFFERPOOL--bufferpool-name-------------------------------------------------------------------+
     +-OVERHEAD--number-of-milliseconds--------------------------------------------------------------+
     +-TRANSFERRATE--number-of-milliseconds----------------------------------------------------------+
     +-+-FILE SYSTEM CACHING----+--------------------------------------------------------------------+
     | '-NO FILE SYSTEM CACHING-'                                                                    |
     +-DROPPED TABLE RECOVERY--+-ON--+---------------------------------------------------------------+
     |                         '-OFF-'                                                               |
     +-SWITCH ONLINE---------------------------------------------------------------------------------+
     +-AUTORESIZE--+-NO--+---------------------------------------------------------------------------+
     |             '-YES-'                                                                           |
     +-INCREASESIZE--integer--+-PERCENT-+------------------------------------------------------------+
     |                        '-+-K-+---'                                                            |
     |                          +-M-+                                                                |
     |                          '-G-'                                                                |
     +-MAXSIZE--+-integer--+-K-+-+-------------------------------------------------------------------+
     |          |          +-M-+ |                                                                   |
     |          |          '-G-' |                                                                   |
     |          '-NONE-----------'                                                                   |
     '-CONVERT TO LARGE------------------------------------------------------------------------------'
```

database-container-clause:

```
          .-,---------------------------------------------------.
          V                                                     |
|--(--+-FILE---+--'container-string'--+-number-of-pages-----+-+--)--------------|
      '-DEVICE-'                      '-integer--+-K-+------'
                                                 +-M-+
                                                 '-G-'
```

drop-container-clause:

```
          .-,---------------------------------.
          V                                   |
|--(--+-FILE---+--'container-string'--+--)--------------------------------|
      '-DEVICE-'
```

system-container-clause:

```
          .-,-------------------.
          V                     |
|--(----'container-string'--+--)------------------------------------------|
```

on-db-partitions-clause:

```
|--ON--+-DBPARTITIONNUM--+----------------------------------------------->
       '-DBPARTITIONNUMS-'
```

```
    ┌─,──────────────────────────────────────────────┐
    ▼                                                │
►─(───db-partition-number1─┬──────────────────────────┬┴──)──────────────────────┤
                            └─TO──db-partition-number2─┘
```

all-containers-clause:

```
            ┌─CONTAINERS─┐
├──(──ALL──┴────────────┴──┬─number-of-pages─────┬──)──────────────────────────┤
                           └─integer──┬─K─┬──────┘
                                      ├─M─┤
                                      └─G─┘
```

Description:

tablespace-name
Names the table space. This is a one-part name. It is a long SQL identifier (either ordinary or delimited).

ADD
Specifies that one or more new containers are to be added to the table space.

TO STRIPE SET *stripeset*
Specifies that one or more new containers are to be added to the table space, and that they will be placed into the given stripe set.

BEGIN NEW STRIPE SET
Specifies that a new stripe set is to be created in the table space, and that one or more containers are to be added to this new stripe set. Containers that are subsequently added using the ADD option will be added to this new stripe set unless TO STRIPE SET is specified.

DROP
Specifies that one or more containers are to be dropped from the table space.

EXTEND
Specifies that existing containers are to be increased in size. The size specified is the size by which the existing container is increased. If the *all-containers-clause* is specified, all containers in the table space will increase by this size.

REDUCE
Specifies that existing containers are to be reduced in size. The size specified is the size by which the existing container is decreased. If the *all-containers-clause* is specified, all containers in the table space will decrease by this size.

RESIZE
Specifies that the size of existing containers is to be changed. The size specified is the new size for the container. If the *all-containers-clause* is specified, all containers in the table space will be changed to this size. If the operation affects more than one container, these containers must all either increase in size, or decrease in size. It is not possible to increase some while decreasing others (SQLSTATE 429BC).

database-container-clause
Adds one or more containers to a DMS table space. The table space must identify a DMS table space that already exists at the application server.

drop-container-clause
Drops one or more containers from a DMS table space. The table space must identify a DMS table space that already exists at the application server.

system-container-clause
Adds one or more containers to an SMS table space on the specified database partitions. The table space must identify an SMS table space that already exists at the application server. There must not be any containers on the specified database partitions for the table space (SQLSTATE 42921).

on-db-partitions-clause
Specifies one or more database partitions for the corresponding container operations.

all-containers-clause
Extends, reduces, or resizes all of the containers in a DMS table space. The table space must identify a DMS table space that already exists at the application server.

PREFETCHSIZE
Specifies to read in data needed by a query prior to it being referenced by the query, so that the query need not wait for I/O to be performed.

AUTOMATIC
Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by DB2, using the following formula:

```
Prefetch size =
 (number of containers) *
 (number of physical disks per container) *
 (extent size)
```

The number of physical disks per container defaults to 1, unless a value is specified through the DB2_PARALLEL_IO registry variable.

DB2 will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is updated at database start-up.

Automatic updating of the prefetch size can be turned off by specifying a numeric value in the PREFETCHSIZE clause.

number-of-pages
Specifies the number of PAGESIZE pages that will be read from the table space when data prefetching is being performed. The prefetch size value can also be specified as an integer value followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

BUFFERPOOL *bufferpool-name*
The name of the buffer pool used for tables in this table space. The buffer pool must currently exist in the database (SQLSTATE 42704). The database partition group of the table space must be defined for the bufferpool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds*
Any numeric literal (integer, decimal, or floating point) that specifies the I/O controller overhead and disk seek and latency time, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

TRANSFERRATE *number-of-milliseconds*
Any numeric literal (integer, decimal, or floating point) that specifies the time to read one page (4K or 8K) into memory, in milliseconds. The number should be an average for all containers that belong to the table space, if not the same for all containers. This value is used to determine the cost of I/O during query optimization.

FILE SYSTEM CACHING or **NO FILE SYSTEM CACHING**
Specifies whether or not I/O operations will be cached at the file system level. Connections to the database must be terminated before a new caching policy takes effect.

FILE SYSTEM CACHING
All I/O operations in the target table space will be cached at the file system level.

NO FILE SYSTEM CACHING
All I/O operations will bypass the file system level cache.

DROPPED TABLE RECOVERY
Specifies whether or not tables that have been dropped from *tablespace-name* can be recovered using the RECOVER DROPPED TABLE ON option of the ROLLFORWARD DATABASE command. For partitioned tables, dropped table recovery is always on, even if dropped table recovery is turned off for non-partitioned tables in one or more table spaces.

ON
Specifies that dropped tables can be recovered.

OFF
Specifies that dropped tables cannot be recovered.

SWITCH ONLINE
Specifies that table spaces in OFFLINE state are to be brought online if their containers have become accessible. If the containers are not accessible, an error is returned (SQLSTATE 57048).

AUTORESIZE
Specifies whether or not the auto-resize capability of a database managed space (DMS) table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full.

NO
Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled. If the auto-resize capability is disabled, any values that have been previously specified for INCREASESIZE or MAXSIZE will not be kept.

YES
Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INCREASESIZE *integer* **PERCENT** or **INCREASESIZE** *integer* **K | M | G**
Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by:

- PERCENT to specify the amount as a percentage of the table space size at the time that a request for space is made. When PERCENT is specified, the integer value must be between 0 and 100 (SQLSTATE 42615).

- K (for kilobytes), M (for megabytes), or G (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

MAXSIZE *integer* **K | M | G** or **MAXSIZE NONE**
Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased.

integer
Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

NONE
Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size (described in “SQL limits”).

CONVERT TO LARGE
Modifies an existing regular DMS table space to be a large DMS table space. The table space and its contents are locked during conversion. This option can only be used on regular DMS table spaces. If an SMS table space, a temporary table space, or the system catalog table space is specified, an error is returned (SQLSTATE 560CF). You cannot convert a table space that contains a data partition of a partitioned table that has data partitions in another table space (SQLSTATE 560CF). Conversion cannot be reversed after being committed. If tables in the table space are defined with DATA CAPTURE CHANGES, consider the storage and capacity limits of the target table and table space.

Rules:

- The BEGIN NEW STRIPE SET clause cannot be specified in the same statement as ADD, DROP, EXTEND, REDUCE, and RESIZE, unless those clauses are being directed to different database partitions (SQLSTATE 429BC).
- The stripe set value specified with the TO STRIPE SET clause must be within the valid range for the table space being altered (SQLSTATE 42615).
- When adding or removing space from the table space, the following rules must be followed:
  – EXTEND and RESIZE can be used in the same statement, provided that the size of each container is increasing (SQLSTATE 429BC).
  – REDUCE and RESIZE can be used in the same statement, provided that the size of each container is decreasing (SQLSTATE 429BC).
  – EXTEND and REDUCE cannot be used in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
  – ADD cannot be used with REDUCE or DROP in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
  – DROP cannot be used with EXTEND or ADD in the same statement, unless they are being directed to different database partitions (SQLSTATE 429BC).
- The AUTORESIZE, INCREASESIZE, or MAXSIZE clause cannot be specified for system managed space (SMS) table spaces, temporary table spaces that were created using automatic storage, or DMS table spaces that are defined to use raw device containers (SQLSTATE 42601).

- The INCREASESIZE or MAXSIZE clause cannot be specified if the table space is not auto-resizable (SQLSTATE 42601).
- When specifying a new maximum size for a table space, the value must be larger than the current size on each database partition (SQLSTATE 560B0).
- Container operations (ADD, EXTEND, RESIZE, REDUCE, DROP, or BEGIN STRIPE SET) cannot be performed on automatic storage table spaces, because the database manager is controlling the space management of such table spaces (SQLSTATE 42858).
- Raw device containers cannot be added to an auto-resizable DMS table space (SQLSTATE 42601).
- The CONVERT TO LARGE clause cannot be specified in the same statement as any other clause (SQLSTATE 429BC).

Notes:

- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container names for the table space cannot exceed 20 480 bytes (SQLSTATE 54034).
- Default container operations are container operations that are specified in the ALTER TABLESPACE statement, but that are not explicitly directed to a specific database partition. These container operations are sent to any database partition that is not listed in the statement. If these default container operations are not sent to any database partition, because all database partitions are explicitly mentioned for a container operation, a warning is returned (SQLSTATE 1758W).
- Once space has been added or removed from a table space, and the transaction is committed, the contents of the table space may be rebalanced across the containers. Access to the table space is not restricted during rebalancing.
- If the table space is in OFFLINE state and the containers have become accessible, the user can disconnect all applications and connect to the database again to bring the table space out of OFFLINE state. Alternatively, SWITCH ONLINE option can bring the table space up (out of OFFLINE) while the rest of the database is still up and being used.
- If adding more than one container to a table space, it is recommended that they be added in the same statement so that the cost of rebalancing is incurred only once. An attempt to add containers to the same table space in separate ALTER TABLESPACE statements within a single transaction will result in an error (SQLSTATE 55041).
- Any attempts to extend, reduce, resize, or drop containers that do not exist will raise an error (SQLSTATE 428B2).
- When extending, reducing, or resizing a container, the container type must match the type that was used when the container was created (SQLSTATE 428B2).
- An attempt to change container sizes in the same table space, using separate ALTER TABLESPACE statements but within a single transaction, will raise an error (SQLSTATE 55041).
- In a partitioned database if more than one database partition resides on the same physical node, the same device or specific path cannot be specified for such database partitions (SQLSTATE 42730). For this environment, either specify a unique *container-string* for each database partition or use a relative path name.
- Although the table space definition is transactional and the changes to the table space definition are reflected in the catalog tables on commit, the buffer pool with the new definition cannot be used until the next time the database is

started. The buffer pool in use, when the ALTER TABLESPACE statement was issued, will continue to be used in the interim.

- ***Conversion to large DMS table spaces:***
  - After conversion, it is recommended that you issue the COMMIT statement and then increase the storage capacity of the table space.
    - If the table space is enabled for auto-resize, the MAXSIZE table space attribute should be increased, unless it is already set to NONE.
    - If the table space is not enabled for auto-resize:
      - Enable auto-resize by issuing the ALTER TABLESPACE statement with the AUTORESIZE YES option, or
      - Add more storage by adding stripe sets, extending the size of existing containers, or both
  - Indexes for tables in a converted table space must be reorganized or rebuilt before they can support large record identifiers (RIDs).
    - The indexes can be reorganized using the REORG INDEXES ALL command (without the CLEANUP ONLY clause). Specify the ALLOW NO ACCESS option for partitioned tables.
    - Alternatively, the tables can be reorganized (not INPLACE), which will rebuild all indexes and enable the tables to support more than 255 rows per page.
    - Any rebuilt Type 1 index is automatically converted to a Type 2 index.
  - To determine which tables do not yet support large RIDs, use the ADMIN_GET_TAB_INFO table function.
- ***Compatibilities***
  - For compatibility with versions earlier than Version 8, the keyword:
    - NODE can be substituted for DBPARTITIONNUM
    - NODES can be substituted for DBPARTITIONNUMS

Examples:

Example 1: Add a device to the PAYROLL table space.

```
ALTER TABLESPACE PAYROLL
  ADD (DEVICE '/dev/rhdisk9' 10000)
```

Example 2: Change the prefetch size and I/O overhead for the ACCOUNTING table space.

```
ALTER TABLESPACE ACCOUNTING
  PREFETCHSIZE 64
  OVERHEAD 19.3
```

Example 3: Create a table space TS1, then resize the containers so that all of the containers have 2000 pages. (Three different ALTER TABLESPACE statements that will accomplish this resizing are shown.)

```
CREATE TABLESPACE TS1
  MANAGED BY DATABASE
  USING (FILE '/conts/cont0' 1000,
         DEVICE '/dev/rcont1' 500,
         FILE 'cont2' 700)
ALTER TABLESPACE TS1
  RESIZE (FILE '/conts/cont0' 2000,
          DEVICE '/dev/rcont1' 2000,
          FILE 'cont2' 2000)
```

OR

```
ALTER TABLESPACE TS1
  RESIZE (ALL 2000)
```

OR

```
ALTER TABLESPACE TS1
  EXTEND (FILE '/conts/cont0' 1000,
          DEVICE '/dev/rcont1' 1500,
          FILE 'cont2' 1300)
```

Example 4: Extend all of the containers in the DATA_TS table space by 1000 pages.

```
ALTER TABLESPACE DATA_TS
  EXTEND (ALL 1000)
```

Example 5: Resize all of the containers in the INDEX_TS table space to 100 megabytes (MB).

```
ALTER TABLESPACE INDEX_TS
  RESIZE (ALL 100 M)
```

Example 6: Add three new containers. Extend the first container, and resize the second.

```
ALTER TABLESPACE TS0
  ADD (FILE 'cont2' 2000, FILE 'cont3' 2000)
  ADD (FILE 'cont4' 2000)
  EXTEND (FILE 'cont0' 100)
  RESIZE (FILE 'cont1' 3000)
```

Example 7: Table space TSO exists on database partitions 0, 1 and 2. Add a new container to database partition 0. Extend all of the containers on database partition 1. Resize a container on all database partitions other than the ones that were explicitly specified (that is, database partitions 0 and 1).

```
ALTER TABLESPACE TS0
  ADD (FILE 'A' 200) ON DBPARTITIONNUM (0)
  EXTEND (ALL 200) ON DBPARTITIONNUM (1)
  RESIZE (FILE 'B' 500)
```

The RESIZE clause is the default container clause in this example, and will be executed on database partition 2, because other operations are being explicitly sent to database partitions 0 and 1. If, however, there had only been these two database partitions, the statement would have succeeded, but returned a warning (SQL1758W) that default containers had been specified but not used.

Example 8: Enable the auto-resize option for table space DMS_TS1, and set its maximum size to 256 megabytes.

```
ALTER TABLESPACE DMS_TS1
  AUTORESIZE YES MAXSIZE 256 M
```

Example 9: Enable the auto-resize option for table space AUTOSTORE1, and change its growth rate to 5%.

```
ALTER TABLESPACE AUTOSTORE1
  AUTORESIZE YES INCREASESIZE 5 PERCENT
```

Example 10: Change the growth rate for an auto-resizable table space named MY_TS to 512 kilobytes, and set its maximum size to be as large as possible.

```
ALTER TABLESPACE MY_TS
  INCREASESIZE 512 K MAXSIZE NONE
```

Related concepts:
- "Automatic resizing of table spaces" in *Administration Guide: Implementation*
- "Automatic storage databases" in *Administration Guide: Implementation*

Related reference:
- "CREATE TABLESPACE " on page 1021
- "System environment variables" in *Performance Guide*
- "ADMINTABINFO administrative view and ADMIN_GET_TAB_INFO table function - Retrieve size and state information for tables" in *Administrative SQL Routines and Views*
- "SQL and XQuery limits" in *SQL Reference, Volume 1*

ALTER VIEW

The ALTER VIEW statement modifies an existing view by:
- Altering a reference type column to add a scope
- Enabling or disabling a view for use in query optimization

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- ALTERIN privilege on the schema of the view
- Definer of the view to be altered
- CONTROL privilege on the view to be altered
- SYSADM or DBADM authority

To enable or disable a view for use in query optimization, the privileges held by the authorization ID of the statement must also include at least one of the following for each of the tables or underlying tables of views that are referenced in the FROM clause of the view fullselect:
- ALTER privilege on the table
- ALTERIN privilege on the schema of the table
- SYSADM or DBADM authority

Syntax:

```
►►──ALTER VIEW──view-name──────────────────────────────────────────────►
```

```
          ┌─────────────────────────────────────────────────────────────────┐
          ▼        ┌─COLUMN─┐                                                  │
►──┬──ALTER──┴────────┴──column-name──ADD SCOPE──┬─typed-table-name─┬──┴─┬──────────►◄
   │                                            └─typed-view-name──┘    │
   └─┬─ENABLE──┬──QUERY OPTIMIZATION────────────────────────────────────────┘
     └─DISABLE─┘
```

Description:

view-name
: Specifies the view that is to be changed. It must be a view that is described in the catalog.

ALTER COLUMN *column-name*
: Specifies the name of the column that is to be altered. The *column-name* must identify an existing column of the view (SQLSTATE 42703). The name cannot be qualified.

ADD SCOPE
: Adds a scope to an existing reference type column that does not already have a scope defined (SQLSTATE 428DK). The column must not be inherited from a superview (SQLSTATE 428DJ).

  typed-table-name
  : Specifies the name of a typed table. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

  typed-view-name
  : Specifies the name of a typed view. The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

ENABLE QUERY OPTIMIZATION or **DISABLE QUERY OPTIMIZATION**
: Specifies whether or not the view and any associated statistics are to be used to improve the optimization of queries. DISABLE QUERY OPTIMIZATION is the default when a view is created.

  ENABLE QUERY OPTIMIZATION
  : Specifies that the view includes statistics that can be used to improve the optimization of queries that involve this view or queries that include subqueries similar to the fullselect of this view.

  DISABLE QUERY OPTIMIZATION
  : Specifies that the view and any associated statistics are not to be used to improve the optimization of queries.

Rules:

- A view cannot be enabled for query optimization if:
  - The view directly or indirectly references a materialized query table (MQT). Note that an MQT or statistical view can reference a statistical view.
  - It is a typed view

Notes:

- To be considered for optimizing a query, a view:
  - Cannot contain aggregation or distinct operations
  - Cannot contain union, except, or intersect operations

  – Cannot contain scalar aggregate (OLAP) functions
- If a view is altered to disable query optimization, cached query plans that used the view for query optimization are invalidated. If a view is altered to enable query optimization, cached query plans are invalidated if they reference the same tables as the newly enabled view references, either directly or indirectly through other views. The invalidation of these cached query plans results in implicit revalidation that takes the view's changed query optimization property into account.

  The query optimization property for a view has no impact on static embedded SQL statements.

COMMENT

The COMMENT statement adds or replaces comments in the catalog descriptions of various objects.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- Owner of the object (underlying table for column or constraint), as recorded in the OWNER column of the catalog view for the object
- ALTERIN privilege on the schema (applicable only to objects that allow more than one-part names)
- CONTROL privilege on the object (applicable only to index, package, table, or view objects)
- ALTER privilege on the object (applicable only to table objects)
- SECADM authority (applicable only to security label, security label component, or security policy objects)
- SYSADM or DBADM authority

Note that for table space or database partition group, the authorization ID must have SYSCTRL or SYSADM authority.

Syntax:

```
►►──COMMENT ON──┬──┤ objects ├──IS──string-constant──────────────────────────┬──►◄
                │                      ┌─,─────────────────────────────────┐  │
                │                      ▼                                   │  │
                └──┬─table-name─┬──(───column-name──IS──string-constant───┴─)─┘
                   └─view-name──┘
```

objects:

```
├──┬─ALIAS──alias-name──────────────────────────────────────────────┬───────┤
   ├─COLUMN──┬─table-name.column-name─┬───────────────────────────────┤
   │         └─view-name.column-name──┘                               │
   ├─CONSTRAINT──table-name.constraint-name───────────────────────────┤
   ├─FUNCTION──function-name──┬────────────────────────┬──────────────┤
   │                          └─(─┬────────────────┬─)─┘              │
   │                              │ ┌─,─────────┐  │                  │
   │                              └─▼─data-type─┴──┘                  │
   ├─SPECIFIC FUNCTION──specific-name─────────────────────────────────┤
   ├─FUNCTION MAPPING──function-mapping-name──────────────────────────┤
   │                  (1)                                             │
   ├─INDEX──index-name────────────────────────────────────────────────┤
   ├─NICKNAME──nickname───────────────────────────────────────────────┤
   ├─DATABASE PARTITION GROUP──db-partition-group-name────────────────┤
   ├─PACKAGE──┬──────────────┬─package-id──┬──────────────────────┬───┤
   │          └─schema-name.─┘             └─┬─VERSION─┬─version-id─┘ │
   │                                         └─────────┘              │
   ├─PROCEDURE──procedure-name──┬────────────────────────┬────────────┤
   │                            └─(─┬────────────────┬─)─┘            │
   │                                │ ┌─,─────────┐  │                │
   │                                └─▼─data-type─┴──┘                │
   ├─SPECIFIC PROCEDURE──specific-name────────────────────────────────┤
   ├─SCHEMA──schema-name──────────────────────────────────────────────┤
   ├─SECURITY LABEL──sec-label-name───────────────────────────────────┤
   ├─SECURITY LABEL COMPONENT──label-comp-name────────────────────────┤
   ├─SECURITY POLICY──label-pol-name──────────────────────────────────┤
   ├─SERVER──server-name──────────────────────────────────────────────┤
   ├─SERVER OPTION──server-option-name──FOR──┤ remote-server ├────────┤
   ├─TABLE──┬─table-name─┬────────────────────────────────────────────┤
   │        └─view-name──┘                                            │
   ├─TABLESPACE──tablespace-name──────────────────────────────────────┤
   ├─TRIGGER──trigger-name────────────────────────────────────────────┤
   ├─┬─────────────────┬─TYPE──type-name──────────────────────────────┤
   │ │            (2)  │                                              │
   │ └─DISTINCT────────┘                                              │
   ├─TYPE MAPPING──type-mapping-name──────────────────────────────────┤
   ├─WRAPPER──wrapper-name────────────────────────────────────────────┤
   └─XSROBJECT──xsrobject-name────────────────────────────────────────┘
```

remote-server:

```
├──┬─SERVER──server-name──────────────────────────────────────────────────┬──────┤
   └─TYPE──server-type──┬─────────────────────────────────────────────────┤
                        └─VERSION──┤ server-version ├──┬─────────────────┤
                                                       └─WRAPPER──wrapper-name─┘
```

server-version:

```
├──┬─version──┬───────────────────────┬─┬─────────────────────────────────┤
   │          └─.──release──┬─────────┤ │
   │                        └─.──mod──┘ │
   └─version-string-constant────────────┘
```

Notes:

1 *Index-name* can be the name of either an index or an index specification.

2 The keyword DATA can be used as a synonym for DISTINCT.

Description:

ALIAS *alias-name*
Indicates a comment will be added or replaced for an alias. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the alias.

COLUMN *table-name.column-name* or *view-name.column-name*
Indicates that a comment for a column will be added or replaced. The *table-name.column-name* or *view-name.column-name* combination must identify a column and table combination that is described in the catalog (SQLSTATE 42704), but must not identify a global temporary table (SQLSTATE 42995). The comment replaces the value of the REMARKS column of the SYSCAT.COLUMNS catalog view for the row that describes the column.

CONSTRAINT *table-name.constraint-name*
Indicates a comment will be added or replaced for a constraint. The *table-name.constraint-name* combination must identify a constraint and the table that it constrains; they must be described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.TABCONST catalog view for the row that describes the constraint.

FUNCTION
Indicates a comment will be added or replaced for a function. The function instance specified must be a user-defined function or function template described in the catalog.

There are several different ways available to identify the function instance:

FUNCTION *function-name*
Identifies the particular function, and is valid only if there is exactly one function with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

FUNCTION *function-name* **(***data-type***,...)**
Provides the function signature, which uniquely identifies the function to be commented upon. The function selection algorithm is *not* used.

function-name
Gives the function name of the function to be commented upon. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(*data-type***,...)**
Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function for which to add or replace the comment.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0 <n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

(Note that the FOR BIT DATA attribute is not considered part of the signature for matching purposes. So, for example, a CHAR FOR BIT DATA specified in the signature would match a function defined with CHAR only, and vice versa.)

If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*
Indicates that comments will be added or replaced for a function (see FUNCTION for other methods of identifying a function). Identifies the particular user-defined function that is to be commented upon, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to comment on a function that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the function.

FUNCTION MAPPING *function-mapping-name*
Indicates a comment will be added or replaced for a function mapping. The *function-mapping-name* must identify a function mapping that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.FUNCMAPPINGS catalog view for the row that describes the function mapping.

INDEX *index-name*
Indicates a comment will be added or replaced for an index or index specification. The *index-name* must identify either a distinct index or an index specification that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.INDEXES catalog view for the row that describes the index or index specification.

NICKNAME *nickname*
Indicates a comment will be added or replaced for a nickname. The *nickname* must be a nickname that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the nickname.

DATABASE PARTITION GROUP *db-partition-group-name*
Indicates a comment will be added or replaced for a database partition group. The *db-partition-group-name* must identify a distinct database partition group that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.DBPARTITIONGROUPS catalog view for the row that describes the database partition group.

PACKAGE *schema-name.package-id*
Indicates that a comment will be added or replaced for a package. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The schema name and package ID, together with the implicitly or explicitly specified version ID, must identify a package that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.PACKAGES catalog view for the row that describes the package.

VERSION *version-id*
Identifies which package version is to be commented on. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be commented on in one invocation of the COMMENT statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PROCEDURE
Indicates a comment will be added or replaced for a procedure. The procedure instance specified must be a procedure described in the catalog.

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*
Identifies the particular procedure, and is valid only if there is exactly one procedure with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is raised.

PROCEDURE *procedure-name* **(***data-type***,...)**
This is used to provide the procedure signature, which uniquely identifies the procedure to be commented upon.

procedure-name
Gives the procedure name of the procedure to be commented upon. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(*data-type***,...)**
Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position. For federated procedures, the data type must match the local catalog information. The number of data types, and the logical concatenation of the data types is used to identify the specific procedure for which to add or replace the comment.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement or the local catalog information, in the case of a federated procedure.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC PROCEDURE *specific-name*
Indicates that comments will be added or replaced for a procedure (see PROCEDURE for other methods of identifying a procedure). Identifies the particular procedure that is to be commented upon, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

It is not possible to comment on a procedure that is in the SYSIBM, SYSFUN, or SYSPROC schema (SQLSTATE 42832).

The comment replaces the value of the REMARKS column of the SYSCAT.ROUTINES catalog view for the row that describes the procedure.

SCHEMA *schema-name*
Indicates a comment will be added or replaced for a schema. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The

comment replaces the value of the REMARKS column of the SYSCAT.SCHEMATA catalog view for the row that describes the schema.

SECURITY LABEL *sec-label-name*
Indicates that a comment will be added or replaced for the security label named *sec-label-name*. The *sec-label-name* must identify a security label that is described in the catalog for the *label-pol-name* (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYLABELS catalog view for the row that describes the security label.

SECURITY LABEL COMPONENT *label-comp-name*
Indicates that a comment will be added or replaced for the security label component named *label-comp-name*. The *label-comp-name* must identify a security label component that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYLABELCOMPONENTS catalog view for the row that describes the security label component.

SECURITY POLICY *label-pol-name*
Indicates that a comment will be added or replaced for the security policy named *label-pol-name*. The *label-pol-name* must identify a security policy that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SECURITYPOLICIES catalog view for the row that describes the security policy.

SERVER *server-name*
Indicates a comment will be added or replaced for a data source. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVERS catalog view for the row that describes the data source.

SERVER OPTION *server-option-name* **FOR** *remote-server*
Indicates a comment will be added or replaced for a server option.

server-option-name
Identifies a server option. This option must be one that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.SERVEROPTIONS catalog view for the row that describes the server option.

remote-server
Describes the data source to which the *server-option* applies.

SERVER *server-name*
Names the data source to which the *server-option* applies. The *server-name* must identify a data source that is described in the catalog.

TYPE *server-type*
Specifies the type of data source—for example, DB2 Universal Database for OS/390 or Oracle—to which the *server-option* applies. The *server-type* can be specified in either lower- or uppercase; it will be stored in uppercase in the catalog.

VERSION
Specifies the version of the data source identified by *server-name*.

version
Specifies the version number. *version* must be an integer.

release
Specifies the number of the release of the version denoted by *version. release* must be an integer.

mod
Specifies the number of the modification of the release denoted by *release. mod* must be an integer.

version-string-constant
Specifies the complete designation of the version. The *version-string-constant* can be a single value (for example, '8i'); or it can be the concatenated values of *version*, *release*, and, if applicable, *mod* (for example, '8.0.3').

WRAPPER *wrapper-name*
Identifies the wrapper that is used to access the data source referenced by *server-name.*

TABLE *table-name* or *view-name*
Indicates a comment will be added or replaced for a table or view. The *table-name* or *view-name* must identify a table or view (not an alias or nickname) that is described in the catalog (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995). The comment replaces the value for the REMARKS column of the SYSCAT.TABLES catalog view for the row that describes the table or view.

TABLESPACE *tablespace-name*
Indicates a comment will be added or replaced for a table space. The *tablespace-name* must identify a distinct table space that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TABLESPACES catalog view for the row that describes the table space.

TRIGGER *trigger-name*
Indicates a comment will be added or replaced for a trigger. The *trigger-name* must identify a distinct trigger that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TRIGGERS catalog view for the row that describes the trigger.

TYPE *type-name*
Indicates a comment will be added or replaced for a user-defined type. The *type-name* must identify a user-defined type that is described in the catalog (SQLSTATE 42704). If DISTINCT is specified, *type-name* must identify a distinct type that is described in the catalog (SQLSTATE 42704). The comment replaces the value of the REMARKS column of the SYSCAT.DATATYPES catalog view for the row that describes the user-defined type.

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

TYPE MAPPING *type-mapping-name*
Indicates a comment will be added or replaced for a user-defined data type mapping. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.TYPEMAPPINGS catalog view for the row that describes the mapping.

WRAPPER *wrapper-name*
Indicates a comment will be added or replaced for a wrapper. The

wrapper-name must identify a wrapper that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.WRAPPERS catalog view for the row that describes the wrapper.

XSROBJECT *xsrobject-name*
Indicates a comment will be added or replaced for an XSR object. The *xsrobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704). The comment replaces the value for the REMARKS column of the SYSCAT.XSROBJECTS catalog view for the row that describes the XSR object.

IS *string-constant*
Specifies the comment to be added or replaced. The *string-constant* can be any character string constant of up to 254 bytes. (Carriage return and line feed each count as 1 byte.)

table-name|*view-name* **({** *column-name* **IS** *string-constant* **} ...)**
This form of the COMMENT statement provides the ability to specify comments for multiple columns of a table or view. The column names must not be qualified, each name must identify a column of the specified table or view, and the table or view must be described in the catalog. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

A comment cannot be made on a column of an inoperative view (SQLSTATE 51024).

Notes:

- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples:

Example 1: Add a comment for the EMPLOYEE table.

```
COMMENT ON TABLE EMPLOYEE
  IS 'Reflects first quarter reorganization'
```

Example 2: Add a comment for the EMP_VIEW1 view.

```
COMMENT ON TABLE EMP_VIEW1
  IS 'View of the EMPLOYEE table without salary information'
```

Example 3: Add a comment for the EDLEVEL column of the EMPLOYEE table.

```
COMMENT ON COLUMN EMPLOYEE.EDLEVEL
  IS 'highest grade level passed in school'
```

Example 4: Add comments for two different columns of the EMPLOYEE table.

```
COMMENT ON EMPLOYEE
  (WORKDEPT IS 'see DEPARTMENT table for names',
  EDLEVEL IS 'highest grade level passed in school' )
```

Example 5: Pellow wants to comment on the CENTRE function, which he created in his PELLOW schema, using the signature to identify the specific function to be commented on.

```
COMMENT ON FUNCTION CENTRE (INT,FLOAT)
  IS 'Frank''s CENTRE fctn, uses Chebychev method'
```

Example 6: McBride wants to comment on another CENTRE function, which she created in the PELLOW schema, using the specific name to identify the function instance to be commented on:

```
COMMENT ON SPECIFIC FUNCTION PELLOW.FOCUS92 IS
  'Louise''s most triumphant CENTRE function, uses the
       Brownian fuzzy-focus technique'
```

Example 7: Comment on the function ATOMIC_WEIGHT in the CHEM schema, where it is known that there is only one function with that name:

```
COMMENT ON FUNCTION CHEM.ATOMIC_WEIGHT
  IS 'takes atomic nbr, gives atomic weight'
```

Example 8: Eigler wants to comment on the SEARCH procedure, which he created in his EIGLER schema, using the signature to identify the specific procedure to be commented on.

```
COMMENT ON PROCEDURE SEARCH (CHAR,INT)
  IS 'Frank''s mass search and replace algorithm'
```

Example 9: Macdonald wants to comment on another SEARCH function, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be commented on:

```
COMMENT ON SPECIFIC PROCEDURE EIGLER.DESTROY IS
  'Patrick''s mass search and destroy algorithm'
```

Example 10: Comment on the procedure OSMOSIS in the BIOLOGY schema, where it is known that there is only one procedure with that name:

```
COMMENT ON PROCEDURE BIOLOGY.OSMOSIS
  IS 'Calculations modelling osmosis'
```

Example 11: Comment on an index specification named INDEXSPEC.

```
COMMENT ON INDEX INDEXSPEC
  IS 'An index specification that indicates to the optimizer
  that the table referenced by nickname NICK1 has an index.'
```

Example 12: Comment on the wrapper whose default name is NET8.

```
COMMENT ON WRAPPER NET8
  IS 'The wrapper for data sources associated with
      Oracle's Net8 client software.'
```

Example 13: Create a comment on the XML schema HR.EMPLOYEE.

```
COMMENT ON XSROBJECT HR.EMPLOYEE
  IS 'This is the base XML Schema for employee data.'
```

Related concepts:

- "Database authorities" on page 74

CREATE DATABASE PARTITION GROUP

The CREATE DATABASE PARTITION GROUP statement defines a new database partition group within the database, assigns database partitions to the database partition group, and records the database partition group definition in the system catalog.

Invocation:

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax:

```
►►──CREATE DATABASE PARTITION GROUP──db-partition-group-name─────────────────────────►

 ┌─ON ALL DBPARTITIONNUMS──────────────────────────────────────────────────┐
►─┼─────────────────────────────────────────────────────────────────────────┼────────►◄
 │                                  ┌─,──────────────────────────────┐     │
 └─ON──┬─DBPARTITIONNUMS─┬──(──▼─db-partition-number1──┬──────────────────────┬─┴──)─┘
       └─DBPARTITIONNUM──┘                                └─TO──db-partition-number2─┘
```

Description:

db-partition-group-name
: Names the database partition group. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *db-partition-group-name* must not identify a database partition group that already exists in the catalog (SQLSTATE 42710). The *db-partition-group-name* must not begin with the characters 'SYS' or 'IBM' (SQLSTATE 42939).

ON ALL DBPARTITIONNUMS
: Specifies that the database partition group is defined over all database partitions defined to the database (`db2nodes.cfg` file) at the time the database partition group is created.

  If a database partition is added to the database system, the ALTER DATABASE PARTITION GROUP statement should be issued to include this new database partition in a database partition group (including IBMDEFAULTGROUP). Furthermore, the REDISTRIBUTE DATABASE PARTITION GROUP command must be issued to move data to the database partition.

ON DBPARTITIONNUMS
: Specifies the database partitions that are in the database partition group. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

  db-partition-number1
  : Specify a database partition number. (A *node-name* of the form `NODE`*nnnnn* can be specified for compatibility with the previous version.)

  TO *db-partition-number2*
  : Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). All database partitions between and including the specified database partition numbers are included in the database partition group.

Rules:

- Each database partition specified by number must be defined in the `db2nodes.cfg` file (SQLSTATE 42729).
- Each *db-partition-number* listed in the ON DBPARTITIONNUMS clause must be appear at most once (SQLSTATE 42728).

- A valid *db-partition-number* is between 0 and 999 inclusive (SQLSTATE 42729).

Notes:

- This statement creates a distribution map for the database partition group. A distribution map identifier (PMAP_ID) is generated for each distribution map. This information is recorded in the catalog and can be retrieved from SYSCAT.DBPARTITIONGROUPS and SYSCAT.PARTITIONMAPS. Each entry in the distribution map specifies the target database partition on which all rows that are hashed reside. For a single-partition database partition group, the corresponding distribution map has only one entry. For a multiple partition database partition group, the corresponding distribution map has 4 096 entries, where the database partition numbers are assigned to the map entries in a round-robin fashion, by default.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
    - NODES can be specified in place of DBPARTITIONNUMS
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples:

Assume that you have a partitioned database with six database partitions defined as 0, 1, 2, 5, 7, and 8.

- Assume that you want to create a database partition group called MAXGROUP on all six database partitions. The statement is as follows:
```
CREATE DATABASE PARTITION GROUP MAXGROUP ON ALL DBPARTITIONNUMS
```
- Assume that you want to create a database partition group called MEDGROUP on database partitions 0, 1, 2, 5, and 8. The statement is as follows:
```
CREATE DATABASE PARTITION GROUP MEDGROUP
  ON DBPARTITIONNUMS( 0 TO 2, 5, 8)
```
- Assume that you want to create a single-partition database partition group MINGROUP on database partition 7. The statement is as follows:
```
CREATE DATABASE PARTITION GROUP MINGROUP
  ON DBPARTITIONNUM (7)
```

Related concepts:

- Chapter 9, “Database partitioning across multiple database partitions,” on page 49

CREATE FUNCTION

The CREATE FUNCTION statement is used to register or define a user-defined function or function template at the current server.

There are five different types of functions that can be created using this statement. Each of these is described separately.

- External Scalar. The function is written in a programming language and returns a scalar value. The external executable is registered in the database, along with various attributes of the function.

- External Table. The function is written in a programming language and returns a complete table. The external executable is registered in the database along with various attributes of the function.
- OLE DB External Table. A user-defined OLE DB external table function is registered in the database to access data from an OLE DB provider.
- Sourced or Template. A source function is implemented by invoking another function (either built-in, external, SQL, or source) that is already registered in the database.

  It is possible to create a partial function, called a *function template*, which defines what types of values are to be returned, but which contains no executable code. The user maps it to a data source function within a federated system, so that the data source function can be invoked from a federated database. A function template can be registered only with an application server that is designated as a federated server.
- SQL Scalar, Table or Row. The function body is written in SQL and defined together with the registration in the database. It returns a scalar value, a table, or a single row.

Related reference:
- "CREATE FUNCTION (External Scalar) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Table) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (OLE DB External Table) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (Sourced or Template) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (SQL Scalar, Table, or Row) statement" in *SQL Reference, Volume 2*

Related samples:
- "dbinline.sqc -- How to use inline SQL Procedure Language (C)"
- "udfcli.sqc -- Call a variety of types of user-defined functions (C)"
- "udfemcli.sqc -- Call a variety of types of embedded SQL user-defined functions. (C)"
- "udfcli.c -- How to work with different types of user-defined functions (UDFs)"
- "udfcli.sqC -- Call a variety of types of user-defined functions (C++)"
- "udfemcli.sqC -- Call a variety of types of embedded SQL user-defined functions. (C++)"
- "UDFCreate.db2 -- How to catalog the Java UDFs contained in UDFsrv.java "
- "UDFjCreate.db2 -- How to catalog the Java UDFs contained in UDFjsrv.java "

CREATE INDEX

The CREATE INDEX statement is used to:
- Define an index on a DB2 table. An index can be defined on XML data, or on relational data.
- Create an index specification (metadata that indicates to the optimizer that a data source table has an index)

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

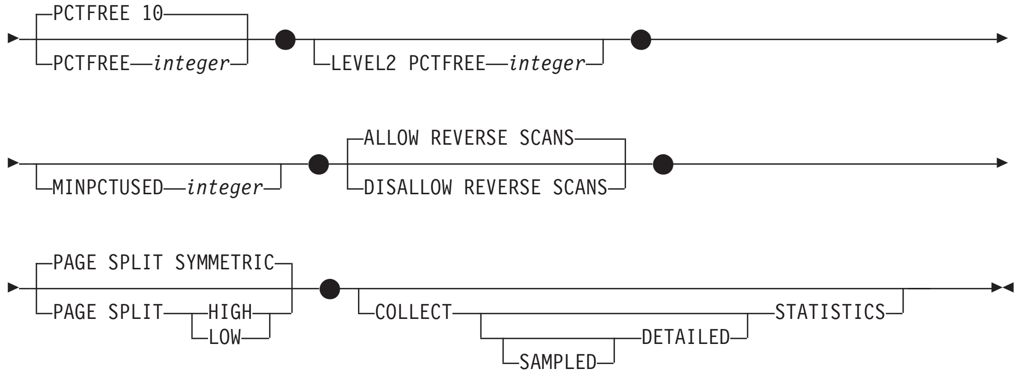
- One of:
  - CONTROL privilege on the table or nickname on which the index is defined
  - INDEX privilege on the table or nickname on which the index is defined

  and one of:
  - IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the index does not exist
  - CREATEIN privilege on the schema, if the schema name of the index refers to an existing schema
- SYSADM or DBADM authority

No explicit privilege is required to create an index on a declared temporary table.

Syntax:

```
>>--CREATE--+--------+--INDEX--index-name------------------------------------->
            '-UNIQUE-'

                                          .-,-----------------------.
                   (1)                    V              .-ASC--.   |
>--ON--+-table-name--------+--(----column-name--+------+-+--)--+-----------------+-->
       |           (2)     |                    '-DESC-'       '-NOT PARTITIONED-'
       '-nickname----------'

>--+----------------------+--+--------------------+--*-------------------------->
   '-IN--tablespace-name--'  '-SPECIFICATION ONLY-'

>--+----------------------------------------+--*-------------------------------->
   |                 .-,-------------.      |
   |          (3)    V               |      |
   '-INCLUDE---------(----column-name-+--)--'

>--+-------------------------------------------------------------------+--*---->
   |                           (4)                                     |
   +-xml-index-specification-------------------------------------------+
   +-CLUSTER-----------------------------------------------------------+
   '-EXTEND USING--index-extension-name--+----------------------------+-'
                                         |    .-,-------------------. |
                                         |    V                     | |
                                         '-(----constant-expression-+--)-'
```



Notes:

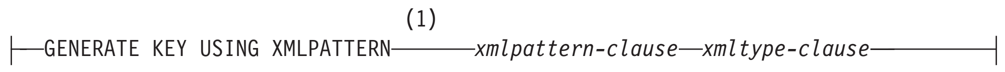
1 In a federated system, *table-name* must identify a table in the federated database. It cannot identify a data source table.

2 If *nickname* is specified, the CREATE INDEX statement creates an index specification. In this case, INCLUDE, CLUSTER, EXTEND USING, PCTFREE, MINPCTUSED, DISALLOW REVERSE SCANS, ALLOW REVERSE SCANS, PAGE SPLIT, or COLLECT STATISTICS cannot be specified.

3 The INCLUDE clause can only be specified if UNIQUE is specified.

4 If *xml-index-specification* is specified, *column-name* DESC, INCLUDE, or CLUSTER cannot be specified.

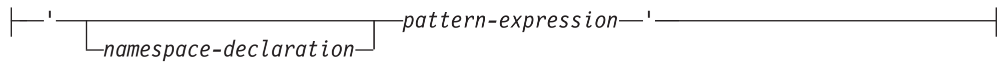
xml-index-specification:


GENERATE KEY USING XMLPATTERN
(1)
xmlpattern-clause—xmltype-clause

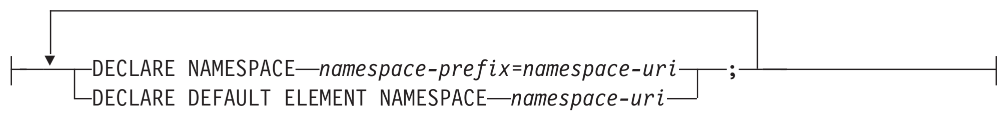
Notes:

1 The alternative syntax GENERATE KEYS USING XMLPATTERN can be used.

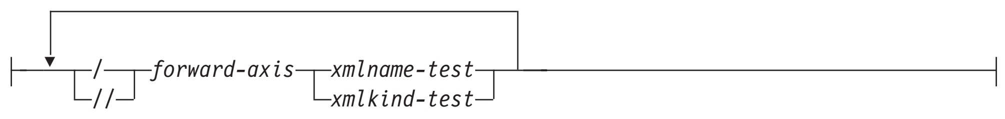
xmlpattern-clause:



namespace-declaration:


DECLARE NAMESPACE—namespace-prefix=namespace-uri
DECLARE DEFAULT ELEMENT NAMESPACE—namespace-uri
;

pattern-expression:



forward-axis:

```
   ┌─child::──────────────┐
├──┼──────────────────────┼──────────────────────────────────────────┤
   ├─@────────────────────┤
   ├─attribute::──────────┤
   ├─descendant::─────────┤
   ├─self::───────────────┤
   └─descendant-or-self::─┘
```

xmlname-test:

```
├──┬─xml-qname────┬──────────────────────────────────────────────────┤
   └─xml-wildcard─┘
```

xml-wildcard:

```
├──┬─*──────────────┬────────────────────────────────────────────────┤
   ├─xml-nsprefix:*─┤
   └─*:xml-ncname───┘
```

xmlkind-test:

```
├──┬─node()───────────────────┬──────────────────────────────────────┤
   ├─text()───────────────────┤
   ├─comment()────────────────┤
   └─processing instruction()─┘
```

xmltype-clause:

```
├──AS──data-type─────────────────────────────────────────────────────┤
```

data-type:

```
├──sqldata-type──────────────────────────────────────────────────────┤
```

sql-data-type:

```
├──SQL──┬─VARCHAR──┬─(──integer──)─┬─┬───────────────────────────────┤
        │          └─HASHED────────┘ │
        ├─DOUBLE─────────────────────┤
        ├─DATE───────────────────────┤
        └─TIMESTAMP──────────────────┘
```

Description:

UNIQUE
If ON *table-name* is specified, UNIQUE prevents the table from containing two or more rows with the same value of the index key. The uniqueness is enforced at the end of the SQL statement that updates rows or inserts new rows.

The uniqueness is also checked during the execution of the CREATE INDEX statement. If the table already contains rows with duplicate key values, the index is not created.

If the index is on an XML column (the index is an index over XML data), the uniqueness applies to values with the specified *pattern-expression* for all rows of the table. Uniqueness is enforced on each value after the value has been converted to the specified *sql-data-type*. Because converting to the specified *sql-data-type* might result in a loss of precision or range, or different values might be hashed to the same key value, multiple values that appear to be unique in the XML document might result in duplicate key errors. The uniqueness of character strings depends on XQuery semantics where trailing blanks are significant. Therefore, values that would be duplicates in SQL but differ in trailing blanks are considered unique values in an index over XML data.

When UNIQUE is used, null values are treated as any other values. For example, if the key is a single column that may contain null values, that column may contain no more than one null value.

If the UNIQUE option is specified, and the table has a distribution key, the columns in the index key must be a superset of the distribution key. That is, the columns specified for a unique index key must include all the columns of the distribution key (SQLSTATE 42997).

Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE).

If ON *nickname* is specified, UNIQUE should be specified only if the data for the index key contains unique values for every row of the data source table. The uniqueness will not be checked.

For an index over XML data, UNIQUE can be specified only if the specified *pattern-expression* specifies a single complete path and does not contain a descendant or descendant-or-self axis, "//", an *xml-wildcard*, *node()*, or *processing-instruction()* (SQLSTATE 429BS).

INDEX *index-name*
Names the index or index specification. The name, including the implicit or explicit qualifier, must not identify an index or index specification that is described in the catalog, or an existing index on a declared temporary table (SQLSTATE 42704). The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

The implicit or explicit qualifier for indexes on declared global temporary tables must be SESSION (SQLSTATE 428EK).

ON *table-name* **or** *nickname*
The *table-name* identifies a table on which an index is to be created. The table must be a base table (not a view), a materialized query table described in the catalog, or a declared temporary table. The name of a declared temporary table must be qualified with SESSION. The *table-name* must not identify a catalog table (SQLSTATE 42832). If UNIQUE is specified and *table-name* is a typed table, it must not be a subtable (SQLSTATE 429B3).

nickname is the nickname on which an index specification is to be created. The *nickname* references either a data source table whose index is described by the index specification, or a data source view that is based on such a table. The *nickname* must be listed in the catalog.

column-name
For an index, *column-name* identifies a column that is to be part of the index

key. For an index specification, *column-name* is the name by which the federated server references a column of a data source table.

Each *column-name* must be an unqualified name that identifies a column of the table. Up to 64 columns can be specified. If *table-name* is a typed table, up to 63 columns can be specified. If *table-name* is a subtable, at least one *column-name* must be introduced in the subtable; that is, not inherited from a supertable (SQLSTATE 428DS). No *column-name* can be repeated (SQLSTATE 42711).

The sum of the stored lengths of the specified columns must not be greater than the index key length limit for the page size. For key length limits, see "SQL limits". If *table-name* is a typed table, the index key length limit is further reduced by 4 bytes. Note that this length limit can be reduced even more by system overhead, which varies according to the data type of the column and whether or not the column is nullable. For more information on overhead affecting this limit, see "Byte Counts" in "CREATE TABLE".

Note that this length can be reduced by system overhead, which varies according to the data type of the column and whether it is nullable. For more information on overhead affecting this limit, see "Byte Counts" in "CREATE TABLE".

No LOB column, DATALINK column, LONG VARCHAR column, LONG VARGRAPHIC column, or distinct type column based on a LOB, DATALINK, LONG VARCHAR, or LONG VARGRAPHIC can be used as part of an index, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008). A structured type column can only be specified if the EXTEND USING clause is also specified (SQLSTATE 42962). If the EXTEND USING clause is specified, only one column can be specified, and the type of the column must be a structured type or a distinct type that is not based on a LOB, DATALINK, LONG VARCHAR, or LONG VARGRAPHIC (SQLSTATE 42997).

If an index has only one column, and that column has the XML data type, and the GENERATE KEY USING XMLPATTERN clause is also specified, the index is an index over XML data. A column with the XML data type can be specified only if the GENERATE KEY USING XMLPATTERN clause is also specified (SQLSTATE 42962). If the GENERATE KEY USING XMLPATTERN clause is specified, only one column can be specified, and the type of the column must be XML.

ASC
Specifies that index entries are to be kept in ascending order of the column values; this is the default setting. ASC cannot be specified for indexes that are defined with EXTEND USING (SQLSTATE 42601).

DESC
Specifies that index entries are to be kept in descending order of the column values. DESC cannot be specified for indexes that are defined with EXTEND USING, or if the index is an index over XML data (SQLSTATE 42601).

NOT PARTITIONED
Indicates that a single index should be created that spans all of the data partitions defined for the table. The *table-name* must identify a table defined with data partitions (SQLSTATE 53036).

IN *tablespace-name*
Specifies the table space in which the index is to be created. This clause is only

supported for indexes on partitioned tables. You can specify this clause even if the INDEX IN clause was specified when the table was created. This will override that clause.

The table space specified by *tablespace-name* must be in the same database partition group as the data table spaces for the table.

If the IN clause is not specified, the index is created in the table space that was specified by the INDEX IN clause on the CREATE TABLE statement. If no INDEX IN clause was specified, the table space of the first visible or attached data partition of the table is used. This is the first partition in the list of data partitions that are sorted on the basis of range specifications.

SPECIFICATION ONLY

Indicates that this statement will be used to create an index specification that applies to the data source table referenced by *nickname*. SPECIFICATION ONLY must be specified if *nickname* is specified (SQLSTATE 42601). It cannot be specified if *table-name* is specified (SQLSTATE 42601).

If the index specification applies to an index that is unique, DB2 does not verify that the column values in the remote table are unique. If the remote column values are not unique, queries against the nickname that include the index column might return incorrect data or errors.

This clause cannot be used when creating an index on a declared temporary table (SQLSTATE 42995).

INCLUDE

This keyword introduces a clause that specifies additional columns to be appended to the set of index key columns. Any columns included with this clause are not used to enforce uniqueness. These included columns might improve the performance of some queries through index only access. The columns must be distinct from the columns used to enforce uniqueness (SQLSTATE 42711). UNIQUE must be specified when INCLUDE is specified (SQLSTATE 42613). The limits for the number of columns and sum of the length attributes apply to all of the columns in the unique key and in the index.

This clause cannot be used with declared temporary tables (SQLSTATE 42995).

column-name

Identifies a column that is included in the index but not part of the unique index key. The same rules apply as defined for columns of the unique index key. The keywords ASC or DESC may be specified following the column-name but have no effect on the order.

INCLUDE cannot be specified for indexes that are defined with EXTEND USING, if *nickname* is specified, or if the index is an XML values index (SQLSTATE 42601).

xml-index-specification

Specifies how index keys are generated from XML documents that are stored in an XML column. *xml-index-specification* cannot be specified if there is more than one index column, or if the column does not have the XML data type.

This clause only applies to XML columns (SQLSTATE 429BS).

GENERATE KEY USING XMLPATTERN *xmlpattern-clause*

Specifies the parts of an XML document that are to be indexed. List data type nodes are not supported in the index. If a node is qualified by the *xmlpattern-clause* and an XML schema exists that specifies that the node is a list data type, then the list data type node cannot be indexed (SQLSTATE 23526 for CREATE INDEX statements, or SQLSTATE 23525 for INSERT and UPDATE statements).

xmlpattern-clause
Contains a pattern expression that identifies the nodes that are to be indexed. It consists of an optional *namespace-declaration* and a required *pattern-expression*.

namespace-declaration
If the pattern expression contains qualified names, a *namespace-declaration* must be specified to define namespace prefixes. A default namespace can be defined for unqualified names.

DECLARE NAMESPACE *namespace-prefix*=*namespace-uri*
Maps *namespace-prefix*, which is an NCName, to *namespace-uri*, which is a string literal. The *namespace-declaration* can contain multiple *namespace-prefix*-to-*namespace-uri* mappings. The *namespace-prefix* must be unique within the list of *namespace-declaration* (SQLSTATE 10503).

DECLARE DEFAULT ELEMENT NAMESPACE *namespace-uri*
Declares the default namespace URI for unqualified element names or types. If no default namespace is declared, unqualified names of elements and types are in no namespace. Only one default namespace can be declared (SQLSTATE 10502).

pattern-expression
Specifies the nodes in an XML document that are indexed. The *pattern-expression* can contain pattern-matching characters (*). It is similar to a path expression in XQuery, but supports a subset of the XQuery language that is supported by DB2.

/ (forward slash)
Separates path expression steps.

// (double forward slash)
This is the abbreviated syntax for /descendant-or-self::node()/. You cannot use *// (double forward slash)* if you also specify UNIQUE.

forward-axis

child::
Specifies children of the context node. This is the default, if no other forward axis is specified.

@ Specifies attributes of the context node. This is the abbreviated syntax for attribute::.

attribute::
Specifies attributes of the context node.

descendant::
Specifies the descendants of the context node. You cannot use *descendant::* if you also specify UNIQUE.

self::
Specifies just the context node itself.

descendant-or-self::
Specifies the context node and the descendants of the context node. You cannot use descendant-or-self:: if you also specify UNIQUE.

xmlname-test
Specifies the node name for the step in the path using a qualified XML name (xml-qname) or a wildcard (xml-wildcard).

xml-ncname
An XML name as defined by XML 1.0. It cannot include a colon character.

xml-qname
Specifies a qualified XML name (also known as a QName) that can have two possible forms:

- xml-nsprefix:xml-ncname, where the xml-nsprefix is an xml-ncname that identifies an in-scope namespace
- xml-ncname, which indicates that the default namespace should be applied as the implicit xml-nsprefix

xml-wildcard
Specifies an xml-qname as a wildcard that can have three possible forms:

- * (a single asterisk character) indicates any xml-qname
- *xml-nsprefix*:* indicates any xml-ncname within the specified namespace
- *:*xml-ncname* indicates a specific XML name in any in-scope namespace

You cannot use *xml-wildcard* if you also specify UNIQUE.

xmlkind-test
Use these options to specify what types of nodes you pattern match. The following options are available to you:

node()
Matches any node. You cannot use *node()* if you also specify UNIQUE.

text()
Matches any text node.

comment()
Matches any comment node.

processing-instruction()
Matches any processing instruction node. You cannot use processing-instruction() if you also specify UNIQUE.

xmltype-clause

AS *data-type*
Specifies the data type to which indexed values are converted before they are stored. Values are converted to the index XML data type that corresponds to the specified index SQL data type.

Table 138. Corresponding index data types

Index XML data type	Index SQL data type
xs:string	VARCHAR(*integer*), VARCHAR HASHED
xs:double	DOUBLE
xs:date	DATE
xs:dateTime	TIMESTAMP

For VARCHAR(*integer*) and VARCHAR HASHED, the value is converted to an xs:string value using the XQuery function fn:string. The length attribute of VARCHAR(*integer*) is applied as a constraint to the resulting xs:string value. An index SQL data type of VARCHAR HASHED applies a hash algorithm to the resulting xs:string value to generate a hash code that is inserted into the index.

For indexes using the data types DOUBLE, DATE, and TIMESTAMP, the value is converted to the index XML data type using the XQuery cast expression. Invalid XML values for the target index XML data type are ignored and are not indexed. The value will be inserted into the table, but it will not be inserted into the index. No error or warning is raised since specifying these data types is not considered a constraint on the values. Note that the index can ignore only invalid XML values for the data type. Valid values must conform to the DB2 representation of the value for the index XML data type, or an error will be issued (SQLSTATE 23526, sqlcode -20306).

If the index is unique, the uniqueness of the value is enforced after the value is converted to the indexed type.

data-type

The following data type is supported:

SQL data type (*sql-data-type***)**

sql-data-type

Supported SQL data types are:

VARCHAR(*integer***)**

If this form of VARCHAR is specified, DB2 uses *integer* as a constraint. If document nodes that are to be indexed have values that are longer than *integer*, the documents are not inserted into the table if the index already exists. If the index does not exist, the index is not created. *integer* is a value between 1 and a page size-dependent maximum. Table 139 shows the maximum value for each page size.

Table 139. Maximum length of document nodes by page size

Page size	Maximum length of document node (bytes)
4KB	817
8KB	1841
16KB	3889

Table 139. Maximum length of document nodes by page size (continued)

Page size	Maximum length of document node (bytes)
32KB	7985

XQuery semantics are used for string comparisons, where trailing blanks are significant. This differs from SQL semantics, where trailing blanks are insignificant during comparisons.

VARCHAR HASHED
Specify VARCHAR HASHED to handle indexing of arbitrary length character strings. The length of an indexed string has no limit. DB2 generates an eight-byte hash code over the entire string. Indexes that use these hashed character strings can be used only for equality lookups. XQuery semantics are used for string equality comparisons, where trailing blanks are significant. This differs from SQL semantics, where trailing blanks are insignificant during comparisons. The hash on the string preserves XQuery semantics for equality and not SQL semantics.

DOUBLE
Specifies that the data type DOUBLE is used for indexing numeric values. Unbounded decimal types and 64 bit integers may lose precision when they are stored as a DOUBLE value. The values for DOUBLE may include the special numeric values *NaN*, *INF*, *-INF*, *+0*, and *-0*, even though the SQL data type DOUBLE itself does not support these values.

DATE
Specifies that the data type DATE is used for indexing XML values. Note that the XML schema data type for xs:date allows greater precision than the SQL data type. If an out-of-range value is encountered, an error is returned.

TIMESTAMP
Specifies that the data type TIMESTAMP is used for indexing XML values. Note that the XML schema data type for xs:dateTime allows greater precision than the SQL data type. If an out-of range value is encountered, an error is returned.

CLUSTER
Specifies that the index is the clustering index of the table. The cluster factor of a clustering index is maintained or improved dynamically as data is inserted into the associated table, by attempting to insert new rows physically close to the rows for which the key values of this index are in the same range. Only one clustering index may exist for a table so CLUSTER may not be specified if

it was used in the definition of any existing index on the table (SQLSTATE 55012). A clustering index may not be created on a table that is defined to use append mode (SQLSTATE 428D8).

CLUSTER is disallowed if *nickname* is specified, or if the index is an index over XML data (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995) or range-clustered tables (SQLSTATE 429BG).

EXTEND USING *index-extension-name*
Names the *index-extension* used to manage this index. If this clause is specified, then there must be only one *column-name* specified and that column must be a structured type or a distinct type (SQLSTATE 42997). The *index-extension-name* must name an index extension described in the catalog (SQLSTATE 42704). For a distinct type, the column must exactly match the type of the corresponding source key parameter in the index extension. For a structured type column, the type of the corresponding source key parameter must be the same type or a supertype of the column type (SQLSTATE 428E0).

This clause cannot be used with declared temporary tables (SQLSTATE 42995).

constant-expression
Identifies values for any required arguments for the index extension. Each expression must be a constant value with a data type that exactly matches the defined data type of the corresponding index extension parameters, including length or precision, and scale (SQLSTATE 428E0). This clause must not exceed 32 768 bytes in length in the database code page (SQLSTATE 22001).

PCTFREE *integer*
Specifies what percentage of each index page to leave as free space when building the index. The first entry in a page is added without restriction. When additional entries are placed in an index page at least *integer* percent of free space is left on each page. The value of *integer* can range from 0 to 99. If a value greater than 10 is specified, only 10 percent free space will be left in non-leaf pages. The default is 10.

PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995).

LEVEL2 PCTFREE *integer*
Specifies what percentage of each index level 2 page to leave as free space when building the index. The value of *integer* can range from 0 to 99. If LEVEL2 PCTFREE is not set, a minimum of 10 or PCTFREE percent of free space is left on all non-leaf pages. If LEVEL2 PCTFREE is set, *integer* percent of free space is left on level 2 intermediate pages, and a minimum of 10 or *integer* percent of free space is left on level 3 and higher intermediate pages.

LEVEL2 PCTFREE is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995).

MINPCTUSED *integer*
Indicates whether index leaf pages are merged online, and the threshold for the minimum percentage of space used on an index leaf page. If, after a key is removed from an index leaf page, the percentage of space used on the page is at or below *integer* percent, an attempt is made to merge the remaining keys on this page with those of a neighboring page. If there is sufficient space on one of these pages, the merge is performed and one of the pages is deleted. The value of *integer* can be from 0 to 99. A value of 50 or below is recommended for performance reasons. Specifying this option will have an impact on update

and delete performance. For type 2 indexes, merging is only done during update and delete operations when there is an exclusive table lock. If an exclusive table lock does not exist, keys are marked as pseudo deleted during update and delete operations, and no merging is done. Consider using the CLEANUP ONLY ALL option of REORG INDEXES to merge leaf pages instead of using the MINPCTUSED option of CREATE INDEX.

MINPCTUSED is disallowed if *nickname* is specified (SQLSTATE 42601). This clause cannot be used with declared temporary tables (SQLSTATE 42995).

DISALLOW REVERSE SCANS
Specifies that an index only supports forward scans or scanning of the index in the order that was defined at index creation time.

DISALLOW REVERSE SCANS cannot be specified together with *nickname* (SQLSTATE 42601).

ALLOW REVERSE SCANS
Specifies that an index can support both forward and reverse scans; that is, scanning of the index in the order that was defined at index creation time, and scanning in the opposite order.

ALLOW REVERSE SCANS cannot be specified together with *nickname* (SQLSTATE 42601).

PAGE SPLIT
Specifies an index split behavior. The default is SYMMETRIC.

SYMMETRIC
Specifies that pages are to be split roughly in the middle.

HIGH
Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that increase with each insertion. For details, see "Options on the CREATE INDEX statement".

LOW
Specifies an index page split behavior that uses the space on index pages efficiently when the values of the index keys being inserted follow a particular pattern. For a subset of index key values, the leftmost column or columns of the index must contain the same value, and the rightmost column or columns of the index must contain values that decrease with each insertion. For details, see "Options on the CREATE INDEX statement".

COLLECT STATISTICS
Specifies that basic index statistics are to be collected during index creation.

DETAILED
Specifies that extended index statistics (CLUSTERFACTOR and PAGE_FETCH_PAIRS) are also to be collected during index creation.

SAMPLED
Specifies that sampling can be used when compiling extended index statistics.

Rules:

- The CREATE INDEX statement will fail (SQLSTATE 01550) if attempting to create an index that matches an existing index.

  Two index descriptions are considered duplicates if:
  - the set of columns (both key and include columns) and their order in the index is the same as that of an existing index AND
  - the ordering attributes are the same AND
  - both the previously existing index and the one being created are non-unique OR the previously existing index is unique AND
  - if both the previously existing index and the one being created are unique, the key columns of the index being created are the same or a superset of key columns of the previously existing index.

  For indexes over XML data, the index descriptions are not considered duplicates if the index names are different, even if the indexed XML column, the XML patterns, and the data type, including its options, are identical.
- Unique indexes on system-maintained MQTs are not supported (SQLSTATE 42809).
- The COLLECT STATISTICS options are not supported if a nickname is specified (SQLSTATE 42601).

Notes:

- Indexes over XML data do not support concurrent write access while CREATE INDEX is executing.
- For relational indexes only: Concurrent read/write access to the table is permitted while an index is being created. Once the index has been built, changes that were made to the table during index creation time are forward-fitted to the new index. Write access to the table is then briefly blocked while index creation completes, after which the new index becomes available.

  To circumvent this default behavior, use the LOCK TABLE statement to explicitly lock the table before issuing a CREATE INDEX statement. (The table can be locked in either SHARE or EXCLUSIVE mode, depending on whether read access is to be allowed.)
- If the named table already contains data, CREATE INDEX creates the index entries for it. If the table does not yet contain data, CREATE INDEX creates a description of the index; the index entries are created when data is inserted into the table.
- Once the index is created and data is loaded into the table, it is advisable to issue the RUNSTATS command. The RUNSTATS command updates statistics collected on the database tables, columns, and indexes. These statistics are used to determine the optimal access path to the tables. By issuing the RUNSTATS command, the database manager can determine the characteristics of the new index. If data has been loaded before the CREATE INDEX statement is issued, it is recommended that the COLLECT STATISTICS option on the CREATE INDEX statement be used as an alternative to the RUNSTATS command.
- Creating an index with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- The optimizer can recommend indexes prior to creating the actual index.
- If an index specification is being defined for a data source table that has an index, the name of the index specification does not have to match the name of the index.
- The optimizer uses index specifications to improve access to the data source tables that the specifications apply to.

- ***Compatibilities***
  - For compatibility with DB2 for OS/390:
    - The following syntax is tolerated and ignored:
      - CLOSE
      - DEFINE
      - FREEPAGE
      - GBPCACHE
      - PIECESIZE
      - TYPE 2
      - using-block
    - The following syntax is accepted as the default behavior:
      - COPY NO
      - DEFER NO

Examples:

Example 1: Create an index named UNIQUE_NAM on the PROJECT table. The purpose of the index is to ensure that there are not two entries in the table with the same value for project name (PROJNAME). The index entries are to be in ascending order.

```
CREATE UNIQUE INDEX UNIQUE_NAM
   ON PROJECT(PROJNAME)
```

Example 2: Create an index named JOB_BY_DPT on the EMPLOYEE table. Arrange the index entries in ascending order by job title (JOB) within each department (WORKDEPT).

```
CREATE INDEX JOB_BY_DPT
   ON EMPLOYEE (WORKDEPT, JOB)
```

Example 3: The nickname EMPLOYEE references a data source table called CURRENT_EMP. After this nickname was created, an index was defined on CURRENT_EMP. The columns chosen for the index key were WORKDEBT and JOB. Create an index specification that describes this index. Through this specification, the optimizer will know that the index exists and what its key is. With this information, the optimizer can improve its strategy to access the table.

```
CREATE UNIQUE INDEX JOB_BY_DEPT
  ON EMPLOYEE (WORKDEPT, JOB)
  SPECIFICATION ONLY
```

Example 4: Create an extended index type named SPATIAL_INDEX on a structured type column location. The description in index extension GRID_EXTENSION is used to maintain SPATIAL_INDEX. The literal is given to GRID_EXTENSION to create the index grid size.

```
CREATE INDEX SPATIAL_INDEX ON CUSTOMER (LOCATION)
  EXTEND USING (GRID_EXTENSION (x'000100100010001000400010'))
```

Example 5: Create an index named IDX1 on a table named TAB1, and collect basic index statistics on index IDX1.

```
CREATE INDEX IDX1 ON TAB1 (col1) COLLECT STATISTICS
```

Example 6: Create an index named IDX2 on a table named TAB1, and collect detailed index statistics on index IDX2.

```
CREATE INDEX IDX2 ON TAB1 (col2) COLLECT DETAILED STATISTICS
```

Example 7: Create an index named IDX3 on a table named TAB1, and collect detailed index statistics on index IDX3 using sampling.

```
CREATE INDEX IDX3 ON TAB1 (col3) COLLECT SAMPLED DETAILED STATISTICS
```

Example 8: Create a unique index named A_IDX on a partitioned table named MYNUMBERDATA in table space IDX_TBSP.

```
CREATE UNIQUE INDEX A_IDX ON MYNUMBERDATA (A) IN IDX_TBSP
```

Example 9: Create a non-unique index named B_IDX on a partitioned table named MYNUMBERDATA in table space IDX_TBSP.

```
CREATE INDEX B_IDX ON MYNUMBERDATA (B)
  NOT PARTITIONED IN IDX_TBSP
```

Example 10: Create an index over XML data on a table named COMPANYINFO, which contains an XML column named COMPANYDOCS. The XML column COMPANYDOCS contains a large number of XML documents similar to the one below:

```
<company name="Company1">
  <emp id="31201" salary="60000" gender="Female">
    <name>
      <first>Laura</first>
      <last>Brown</last>
    </name>
    <dept id="M25">
      Finance
    </dept>
  </emp>
</company>
```

Users of the COMPANYINFO table often need to retrieve employee information using the employee ID. An index like the following one can make that retrieval more efficient.

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
  GENERATE KEY USING XMLPATTERN '/company/emp/@id'
    AS SQL DOUBLE
```

Example 11: The following index is logically equivalent to the index created in the previous example, except that it uses unabbreviated syntax.

```
CREATE INDEX EMPINDEX ON COMPANYINFO(COMPANYDOCS)
  GENERATE KEY USING XMLPATTERN '/child::company/child::emp/attribute::id'
    AS SQL DOUBLE
```

Example 12: Create an index on a column named DOC, indexing only the book title as a VARCHAR(100). Because the book title should be unique across all books, the index must be unique.

```
CREATE UNIQUE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN '/book/title'
    AS SQL VARCHAR(100)
```

Example 13: Create an index on a column named DOC, indexing the chapter number as a DOUBLE. This example includes namespace declarations.

```
CREATE INDEX MYDOCSIDX ON MYDOCS(DOC)
  GENERATE KEY USING XMLPATTERN
    'declare namespace b="http://www.foobar.com/book/";
      declare namespace c="http://acme.org/chapters";
        /b:book/c:chapter/@number'
    AS SQL DOUBLE
```

Related concepts:

- "Data type conversion for indexes over XML data" in *Performance Guide*
- "Restrictions on indexes over XML data" in *Performance Guide*
- "Understanding clustering index behavior on partitioned tables" in *Performance Guide*
- "Understanding index behavior on partitioned tables" in *Performance Guide*
- "XMLEXISTS predicate usage" in *Performance Guide*
- "Options on the CREATE INDEX statement" on page 316

Related reference:

- "CREATE INDEX EXTENSION statement" in *SQL Reference, Volume 2*
- "CREATE TABLE " on page 956
- "Interaction of triggers and constraints" in *SQL Reference, Volume 1*
- "SQL and XQuery limits" in *SQL Reference, Volume 1*

Related samples:

- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"
- "TbGenCol.java -- How to use generated columns (JDBC)"

CREATE METHOD

The CREATE METHOD statement is used to associate a method body with a method specification that is already part of the definition of a user-defined structured type.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- CREATEIN privilege on the schema of the structured type referred to in the CREATE METHOD statement
- The DEFINER of the structured type referred to in the CREATE METHOD statement
- SYSADM or DBADM authority

To associate an external method body with its method specification, the privileges held by the authorization ID of the statement must also include at least one of the following:

- CREATE_EXTERNAL_ROUTINE authority on the database
- SYSADM or DBADM authority

When creating an SQL method, the privileges held by the authorization ID of the statement must also include, for each table, view, or nickname identified in any fullselect:

- CONTROL privilege on that table, view, or nickname, or
- SELECT privilege on that table, view, or nickname

If the definer of an SQL method can only create the method because the definer has SYSADM authority, the definer is granted implicit DBADM authority for the purpose of creating the method.

Group privileges other than PUBLIC are not considered for any table or view specified in the CREATE METHOD statement.

Authorization requirements of the data source for the table or view referenced by the nickname are applied when the method is invoked. The authorization ID of the connection can be mapped to a different remote authorization ID.

Syntax:

```
>>──CREATE──┬─METHOD──┬─method-name────────────┬─FOR──type-name─┬──────────────────►
            │         └─┤ method-signature ├───┘                │
            └─SPECIFIC METHOD──specific-name────────────────────┘

►──┬─●─EXTERNAL─┬──────────────────────┬──●─┬──────────────────────────────┬─●─┬──────►◄
   │            └─NAME──┬─'string'───┬─┘    └─TRANSFORM GROUP──group-name──┘   │
   │                    └─identifier─┘                                         │
   │ ┌─INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST─┐                            │
   └─┴─INHERIT ISOLATION LEVEL WITH LOCK REQUEST────┴──┤ SQL-method-body ├─────┘
```

method-signature:

```
├──method-name──(──┬──────────────────────────────────────────────────────┬──)─────────►
                   │ ┌─,─────────────────────────────────────────────────┐ │
                   │ ▼                                                   │ │
                   └───┬────────────────┬──data-type1──┬────────────┬────┴─┘
                       └─parameter-name─┘              └─AS LOCATOR─┘

►──┬─────────────────────────────────────────────────────────────────┬─────────────┤
   └─RETURNS──┬─data-type2──┬────────────┬─────────────────────────┬─┘
              │             └─AS LOCATOR─┘                         │
              └─data-type3──CAST FROM──data-type4──┬────────────┬──┘
                                                   └─AS LOCATOR─┘
```

SQL-method-body:

```
├──┬─RETURN Statement──────────────┬──────────────────────────────────────────────┤
   └─dynamic-compound-statement────┘
```

Description:

METHOD
: Identifies an existing method specification that is associated with a user-defined structured type. The method-specification can be identified through one of the following means:

method-name
: Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*). There must be only one method specification for *type-name* that has this *method-name* (SQLSTATE 42725).

method-signature
Provides the method signature which uniquely identifies the method to be defined. The method signature must match the method specification that was provided on the CREATE TYPE or ALTER TYPE statement (SQLSTATE 42883).

method-name
Names the method specification for which a method body is being defined. The implicit schema is the schema of the subject type (*type-name*).

parameter-name
Identifies the parameter name. If parameter names are provided in the method signature, they must be exactly the same as the corresponding parts of the matching method specification. Parameter names are supported in this statement solely for documentation purposes.

data-type1
Specifies the data type of each parameter.

AS LOCATOR
For the LOB types or distinct types which are based on a LOB type, the AS LOCATOR clause can be added.

RETURNS
This clause identifies the output of the method. If a RETURNS clause is provided in the method signature, it must be exactly the same as the corresponding part of the matching method specification on CREATE TYPE. The RETURNS clause is supported in this statement solely for documentation purposes.

data-type2
Specifies the data type of the output.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be added. This indicates that a LOB locator is to be returned by the method instead of the actual value.

data-type3 **CAST FROM** *data-type4*
This form of the RETURNS clause is used to return a different data type to the invoking statement from the data type that was returned by the function code.

AS LOCATOR
For LOB types or distinct types which are based on LOB types, the AS LOCATOR clause can be used to indicate that a LOB locator is to be returned from the method instead of the actual value.

FOR *type-name*
Names the type for which the specified method is to be associated. The name must identify a type already described in the catalog. (SQLSTATE 42704) In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

SPECIFIC METHOD *specific-name*
Identifies the particular method, using the specific name either specified or defaulted to at CREATE TYPE time. The specific-name must identify a method specification in the named or implicit schema; otherwise, an error is raised (SQLSTATE 42704).

EXTERNAL
This clause indicates that the CREATE METHOD statement is being used to register a method, based on code written in an external programming language, and adhering to the documented linkage conventions and interface. The matching method-specification in CREATE TYPE must specify a LANGUAGE other than SQL. When the method is invoked, the subject of the method is passed to the implementation as an implicit first parameter.

If the NAME clause is not specified, "NAME *method-name*" is assumed.

NAME
This clause identifies the name of the user-written code which implements the method being defined.

'*string***'**
The 'string' option is a string constant with a maximum of 254 bytes. The format used for the string is dependent on the LANGUAGE specified. For more information on the specific language conventions, see "CREATE FUNCTION (External Scalar)".

identifier
This identifier specified is an SQL identifier. The SQL identifier is used as the library-id in the string. Unless it is a delimited identifier, the identifier is folded to upper case. If the identifier is qualified with a schema name, the schema name portion is ignored. This form of NAME can only be used with LANGUAGE C (as defined in the method-specification on CREATE TYPE).

TRANSFORM GROUP *group-name*
Indicates the transform group that is used for user-defined structured type transformations when invoking the method. A transform is required since the method definition includes a user-defined structured type.

It is strongly recommended that a transform group name be specified; if this clause is not specified, the default group-name used is DB2_FUNCTION. If the specified (or default) group-name is not defined for a referenced structured type, an error results (SQLSTATE 42741). Likewise, if a required FROM SQL or TO SQL transform function is not defined for the given group-name and structured type, an error results (SQLSTATE 42744).

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST or **INHERIT ISOLATION LEVEL WITH LOCK REQUEST**
Specifies whether or not a lock request can be associated with the isolation-clause of the statement when the method inherits the isolation level of the statement that invokes the method. The default is INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST.

INHERIT ISOLATION LEVEL WITHOUT LOCK REQUEST
Specifies that, as the method inherits the isolation level of the invoking statement, it cannot be invoked in the context of an SQL statement which includes a lock-request-clause as part of a specified isolation-clause (SQLSTATE 42601).

INHERIT ISOLATION LEVEL WITH LOCK REQUEST
Specifies that, as the method inherits the isolation level of the invoking statement, it also inherits the specified lock-request-clause.

SQL-method-body
The SQL-method-body defines how the method is implemented if the method specification in CREATE TYPE is LANGUAGE SQL.

The SQL-method-body must comply with the following parts of method specification:

- DETERMINISTIC or NOT DETERMINISTIC (SQLSTATE 428C2)
- EXTERNAL ACTION or NO EXTERNAL ACTION (SQLSTATE 428C2)
- CONTAINS SQL or READS SQL DATA (SQLSTATE 42985)

Parameter names can be referenced in the SQL-method-body. The subject of the method is passed to the method implementation as an implicit first parameter named SELF.

For additional details, see "Compound SQL (Dynamic)" and "RETURN Statement".

Rules:

- The method specification must be previously defined using the CREATE TYPE or ALTER TYPE statement before CREATE METHOD can be used (SQLSTATE 42723).
- If the method being created is an overriding method, those packages that are dependent on the following methods are invalidated:
  - The original method
  - Other overriding methods that have as their subject a supertype of the method being created
- The XML data type cannot be used in a method.

Notes:

- If the method allows SQL, the external program must not attempt to access any federated objects (SQLSTATE 55047).
- ***Privileges***

  The definer of a method always receives the EXECUTE privilege on the method, as well as the right to drop the method.

  If an EXTERNAL method is created, the definer of the method always receives the EXECUTE privilege WITH GRANT OPTION.

  If an SQL method is created, the definer of the method will only be given the EXECUTE privilege WITH GRANT OPTION on the method when the definer has WITH GRANT OPTION on all privileges required to define the method, or if the definer has SYSADM or DBADM authority. The definer of an SQL method only acquires privileges if the privileges from which they are derived exist at the time the method is created. The definer must have these privileges either directly, or because PUBLIC has the privileges. Privileges held by groups of which the method definer is a member are not considered. When using the method, the connected user's authorization ID must have the valid privileges on the table or view that the nickname references at the data source.
- ***Table access restrictions***

If a method is defined as READS SQL DATA, no statement in the method can access a table that is being modified by the statement which invoked the method (SQLSTATE 57053).

Examples:

Example 1:

```
CREATE METHOD BONUS (RATE DOUBLE)
  FOR EMP
  RETURN SELF..SALARY * RATE
```

Example 2:

```
CREATE METHOD SAMEZIP (addr address_t)
  RETURNS INTEGER
  FOR address_t
  RETURN
    (CASE
      WHEN (self..zip = addr..zip)
        THEN 1
      ELSE 0
    END)
```

Example 3:

```
CREATE METHOD DISTANCE (address_t)
  FOR address_t
  EXTERNAL NAME 'addresslib!distance'
  TRANSFORM GROUP func_group
```

Related reference:

- "Compound SQL (Dynamic) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (External Scalar) statement" in *SQL Reference, Volume 2*
- "RETURN statement" in *SQL Reference, Volume 2*

CREATE PROCEDURE (SQL)

The CREATE PROCEDURE (SQL) statement defines an SQL procedure at the current server.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- BINDADD privilege on the database, and one of the following:
  - IMPLICIT_SCHEMA privilege on the database, if the implicit or explicit schema name of the procedure does not exist
  - CREATEIN privilege on the schema, if the schema name of the procedure refers to an existing schema
- SYSADM or DBADM authority

If a procedure definer can only create the procedure because the definer has SYSADM authority, the definer is granted implicit DBADM authority for the purpose of creating the procedure.

If the authorization ID of the statement does not have SYSADM or DBADM authority, the privileges held by the authorization ID of the statement must also include all of the privileges necessary to invoke the SQL statements that are specified in the procedure body.

Syntax:

```
►►──CREATE PROCEDURE──procedure-name──────────────────────────────────────────►

►──┬───────────────────────────────────────────────┬──●────────────────────────►
   └─(──┬───────────────────────────────────────┬─)─┘
        │  ┌─,────────────────────────────────┐ │
        │  ▼ ┌─IN────┐                        │ │
        └────┼───────┼──parameter-name──data-type─┴─┘
             ├─OUT───┤
             └─INOUT─┘

                                        ┌─DYNAMIC RESULT SETS 0────────┐
►──┬─────────────────────────────┬──●──┼──────────────────────────────┼──●────►
   └─SPECIFIC──specific-name─────┘     └─DYNAMIC RESULT SETS──integer─┘

   ┌─MODIFIES SQL DATA─┐     ┌─NOT DETERMINISTIC─┐     ┌─CALLED ON NULL INPUT─┐
►──┼───────────────────┼──●──┼───────────────────┼──●──┴──────────────────────┴──►
   ├─CONTAINS SQL──────┤     └─DETERMINISTIC─────┘
   └─READS SQL DATA────┘

        ┌─INHERIT SPECIAL REGISTERS─┐       ┌─OLD SAVEPOINT LEVEL─┐
►──●────┴───────────────────────────┴────●──┼─────────────────────┼──●────────►
                                            └─NEW SAVEPOINT LEVEL─┘

   ┌─LANGUAGE SQL─┐       ┌─EXTERNAL ACTION────┐
►──┴──────────────┴────●──┼────────────────────┼──●───────────────────────────►
                          └─NO EXTERNAL ACTION─┘

►──┬─────────────────────────────────┬──●──┤ SQL-procedure-body ├──────────►◄
   └─PARAMETER CCSID──┬─ASCII───┬────┘
                      └─UNICODE─┘
```

SQL-procedure-body:

```
├──SQL-procedure-statement─────────────────────────────────────────────────┤
```

Description:

procedure-name
: Names the procedure being defined. It is a qualified or unqualified name that designates a procedure. The unqualified form of *procedure-name* is an SQL identifier (with a maximum length of 128). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option

implicitly specifies the qualifier for unqualified object names. The qualified form is a *schema-name* followed by a period and an SQL identifier.

The name, including the implicit or explicit qualifiers, together with the number of parameters, must not identify a procedure described in the catalog (SQLSTATE 42723). The unqualified name, together with the number of parameters, is unique within its schema, but does not need to be unique across schemas.

If a two-part name is specified, the *schema-name* cannot begin with 'SYS'; otherwise, an error is returned (SQLSTATE 42939).

*(***IN | OUT | INOUT** *parameter-name data-type,...)*
Identifies the parameters of the procedure, and specifies the mode, name, and data type of each parameter. One entry in the list must be specified for each parameter that the procedure will expect.

It is possible to register a procedure that has no parameters. In this case, the parentheses must still be coded, with no intervening data types. For example:

```
CREATE PROCEDURE SUBWOOFER() ...
```

No two identically-named procedures within a schema are permitted to have exactly the same number of parameters. A duplicate signature raises an SQL error (SQLSTATE 42723).

For example, given the statements:

```
CREATE PROCEDURE PART (IN NUMBER INT, OUT PART_NAME CHAR(35)) ...
CREATE PROCEDURE PART (IN COST DECIMAL(5,3), OUT COUNT INT) ...
```

the second statement will fail because the number of parameters in the procedure is the same, even if the data types are not.

IN | OUT | INOUT
Specifies the mode of the parameter.

If an error is returned by the procedure, OUT parameters are undefined and INOUT parameters are unchanged.

IN Identifies the parameter as an input parameter to the procedure. Any changes made to the parameter within the procedure are not available to the calling SQL application when control is returned. The default is IN.

OUT Identifies the parameter as an output parameter for the procedure.

INOUT
Identifies the parameter as both an input and output parameter for the procedure.

parameter-name
Specifies the name of the parameter. The parameter name must be unique for the procedure (SQLSTATE 42734).

data-type
Specifies the data type of the parameter.

- SQL data type specifications and abbreviations that can be specified in the *data-type* definition of a CREATE TABLE statement, and that have a correspondence in the language that is being used to write the procedure, may be specified.
- LONG VARCHAR, LONG VARGRAPHIC, DATALINK, REFERENCE, and user-defined structured types are not supported (SQLSTATE 429BB).

SPECIFIC *specific-name*
Provides a unique name for the instance of the procedure that is being defined. This specific name can be used when dropping the procedure or commenting on the procedure. It can never be used to invoke the procedure. The unqualified form of *specific-name* is an SQL identifier (with a maximum length of 18). The qualified form is a *schema-name* followed by a period and an SQL identifier. The name, including the implicit or explicit qualifier, must not identify another procedure instance that exists at the application server; otherwise an error (SQLSTATE 42710) is raised.

The *specific-name* can be the same as an existing *procedure-name*.

If no qualifier is specified, the qualifier that was used for *procedure-name* is used. If a qualifier is specified, it must be the same as the explicit or implicit qualifier for *procedure-name*, or an error (SQLSTATE 42882) is raised.

If *specific-name* is not specified, a unique name is generated by the database manager. The unique name is 'SQL' followed by a character timestamp: 'SQL*yymmddhhmmssxxx*'.

DYNAMIC RESULT SETS *integer*
Indicates the estimated upper bound of returned result sets for the procedure.

CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA
Indicates the level of data access for SQL statements included in the procedure.

CONTAINS SQL
Indicates that SQL statements that neither read nor modify SQL data can be executed by the procedure (SQLSTATE 38004 or 42985). Statements that are not supported in procedures might return a different error (SQLSTATE 38003 or 42985).

READS SQL DATA
Indicates that some SQL statements that do not modify SQL data can be included in the procedure (SQLSTATE 38002 or 42985). Statements that are not supported in procedures might return a different error (SQLSTATE 38003 or 42985).

MODIFIES SQL DATA
Indicates that the procedure can execute any SQL statement except statements that are not supported in procedures (SQLSTATE 38003 or 42985).

If the BEGIN ATOMIC clause is used in a compound SQL procedure, the procedure can only be created if it is defined as MODIFIES SQL DATA.

DETERMINISTIC or **NOT DETERMINISTIC**
This clause specifies whether the procedure always returns the same results for given argument values (DETERMINISTIC) or whether the procedure depends on some state values that affect the results (NOT DETERMINISTIC). That is, a DETERMINISTIC procedure must always return the same result from successive invocations with identical inputs.

This clause currently does not impact processing of the procedure.

CALLED ON NULL INPUT
CALLED ON NULL INPUT always applies to procedures. This means that the procedure is called regardless of whether any arguments are null. Any OUT or INOUT parameter can return a null value or a normal (non-null) value. Responsibility for testing for null argument values lies with the procedure.

INHERIT SPECIAL REGISTERS
This optional clause specifies that updatable special registers in the procedure will inherit their initial values from the environment of the invoking statement. For a routine invoked in a nested object (for example a trigger or view), the initial values are inherited from the runtime environment (not inherited from the object definition).

No changes to the special registers are passed back to the caller of the procedure.

Non-updatable special registers, such as the datetime special registers, reflect a property of the statement currently executing, and are therefore set to their default values.

OLD SAVEPOINT LEVEL or **NEW SAVEPOINT LEVEL**
Specifies whether or not this procedure establishes a new savepoint level for savepoint names and effects. OLD SAVEPOINT LEVEL is the default behavior. For more information about savepoint levels, see “Rules” in “SAVEPOINT”.

LANGUAGE SQL
This clause is used to specify that the procedure body is written in the SQL language.

EXTERNAL ACTION or **NO EXTERNAL ACTION**
Specifies whether the procedure takes some action that changes the state of an object not managed by the database manager (EXTERNAL ACTION), or not (NO EXTERNAL ACTION). The default is EXTERNAL ACTION. If NO EXTERNAL ACTION is specified, the system can use certain optimizations that assume the procedure has no external impact.

PARAMETER CCSID
Specifies the encoding scheme to use for all string data passed into and out of the procedure. If the PARAMETER CCSID clause is not specified, the default is PARAMETER CCSID UNICODE for Unicode databases, and PARAMETER CCSID ASCII for all other databases.

ASCII
Specifies that string data is encoded in the database code page. If the database is a Unicode database, PARAMETER CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE
Specifies that character data is in UTF-8, and that graphic data is in UCS-2. If the database is not a Unicode database, PARAMETER CCSID UNICODE cannot be specified (SQLSTATE 56031).

SQL-procedure-body
Specifies the SQL statement that is the body of the SQL procedure. Multiple SQL-procedure-statements can be specified within a procedure-compound-statement. See *SQL-procedure-statement* in “Compound SQL (Procedure)”.

Rules:

- A procedure that is called from within a dynamic compound statement will execute as if it were created specifying NEW SAVEPOINT LEVEL, even if OLD SAVEPOINT LEVEL was specified or defaulted to when the procedure was created.

Notes:

- Creating a procedure with a schema name that does not already exist will result in the implicit creation of that schema, provided that the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.

- ***Privileges***

  The definer of a procedure always receives the EXECUTE privilege WITH GRANT OPTION on the procedure, as well as the right to drop the procedure.
- ***Compatibilities***
  - For compatibility with DB2 UDB for OS/390 and z/OS:
    - The following syntax is accepted as the default behavior:
      - ASUTIME NO LIMIT
      - COMMIT ON RETURN NO
      - NO COLLID
      - STAY RESIDENT NO
  - For compatibility with previous versions of DB2:
    - RESULT SETS can be specified in place of DYNAMIC RESULT SETS.
    - NULL CALL can be specified in place of CALLED ON NULL INPUT.

Examples:

Example 1: Create an SQL procedure that returns the median staff salary. Return a result set containing the name, position, and salary of all employees who earn more than the median salary.

```
CREATE PROCEDURE MEDIAN_RESULT_SET (OUT medianSalary DOUBLE)
  RESULT SETS 1
  LANGUAGE SQL
BEGIN
  DECLARE v_numRecords INT DEFAULT 1;
  DECLARE v_counter INT DEFAULT 0;

  DECLARE c1 CURSOR FOR
    SELECT CAST(salary AS DOUBLE)
      FROM staff
      ORDER BY salary;
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, job, CAST(salary AS INTEGER)
      FROM staff
      WHERE salary > medianSalary
      ORDER BY salary;

  DECLARE EXIT HANDLER FOR NOT FOUND
    SET medianSalary = 6666;

  SET medianSalary = 0;
  SELECT COUNT(*) INTO v_numRecords
    FROM STAFF;
  OPEN c1;
  WHILE v_counter < (v_numRecords / 2 + 1)
  DO
    FETCH c1 INTO medianSalary;
    SET v_counter = v_counter + 1;
  END WHILE;
  CLOSE c1;
  OPEN c2;
END
```

Related reference:

- “Special registers” on page 1523
- “SQL statements allowed in routines” in *SQL Reference, Volume 1*

- "Compound SQL (Procedure) statement" in *SQL Reference, Volume 2*
- "SAVEPOINT statement" in *SQL Reference, Volume 2*

Related samples:

- "basecase.db2 -- To create the UPDATE_SALARY SQL procedure "
- "nestcase.db2 -- To create the BUMP_SALARY SQL procedure "
- "nestedsp.db2 -- To create the OUT_AVERAGE, OUT_MEDIAN and MAX_SALARY SQL procedures"
- "rsultset.db2 -- To register and create the MEDIAN_RESULT_SET SQL procedure"

CREATE SCHEMA

The CREATE SCHEMA statement defines a schema. It is also possible to create some objects and grant privileges on objects within the statement.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

An authorization ID that holds SYSADM or DBADM authority can create a schema with any valid *schema-name* or *authorization-name*.

An authorization ID that does not hold SYSADM or DBADM authority can only create a schema with a *schema-name* or *authorization-name* that matches the authorization ID of the statement.

If the statement includes a *schema-SQL-statement*, the privileges held by the *authorization-name* (which, if not specified, defaults to the authorization ID of the statement) must include at least one of the following:

- The privileges required to perform each *schema-SQL-statement*
- SYSADM or DBADM authority

Syntax:

```
>>--CREATE SCHEMA--+-schema-name----------------------------------+-->
                   +-AUTHORIZATION--authorization-name------------+
                   '-schema-name--AUTHORIZATION--authorization-name-'

>--+------------------------------+-----------------------------------><
   |  .------------------------.  |
   |  V                        |  |
   '----schema-SQL-statement---+--'
```

Description:

schema-name
: Names the schema. The name must not identify a schema already described in

the catalog (SQLSTATE 42710). The name cannot begin with 'SYS' (SQLSTATE 42939). The owner of the schema is the authorization ID that issued the statement.

AUTHORIZATION *authorization-name*
Identifies the user who is the owner of the schema. The value of *authorization-name* is also used to name the schema. The *authorization-name* must not identify a schema already described in the catalog (SQLSTATE 42710).

schema-name **AUTHORIZATION** *authorization-name*
Identifies a schema called *schema-name*, whose owner is *authorization-name*. The *schema-name* must not identify a schema already described in the catalog (SQLSTATE 42710). The *schema-name* cannot begin with 'SYS' (SQLSTATE 42939).

schema-SQL-statement
SQL statements that can be included as part of the CREATE SCHEMA statement are:

- CREATE TABLE statement, excluding typed tables and materialized query tables
- CREATE VIEW statement, excluding typed views
- CREATE INDEX statement
- COMMENT statement
- GRANT statement

Notes:

- The owner of the schema is determined as follows:
  - If an AUTHORIZATION clause is specified, the specified *authorization-name* is the schema owner
  - If an AUTHORIZATION clause is not specified, the authorization ID that issued the CREATE SCHEMA statement is the schema owner.
- The schema owner is assumed to be a user (not a group).
- When the schema is explicitly created with the CREATE SCHEMA statement, the schema owner is granted CREATEIN, DROPIN, and ALTERIN privileges on the schema with the ability to grant these privileges to other users.
- The definer of any object created as part of the CREATE SCHEMA statement is the schema owner. The schema owner is also the grantor for any privileges granted as part of the CREATE SCHEMA statement.
- Unqualified object names in any SQL statement within the CREATE SCHEMA statement are implicitly qualified by the name of the created schema.
- If the CREATE statement contains a qualified name for the object being created, the schema name specified in the qualified name must be the same as the name of the schema being created (SQLSTATE 42875). Any other objects referenced within the statements may be qualified with any valid schema name.
- It is recommended not to use ″SESSION″ as a schema name. Since declared temporary tables must be qualified by ″SESSION″, it is possible to have an application declare a temporary table with a name identical to that of a persistent table. An SQL statement that references a table with the schema name ″SESSION″ will resolve (at statement compile time) to the declared temporary table rather than a persistent table with the same name. Since an SQL statement is compiled at different times for static embedded and dynamic embedded SQL statements, the results depend on when the declared temporary table is defined. If persistent tables, views or aliases are not defined with a schema name of ″SESSION″, these issues do not require consideration.

Examples:

Example 1: As a user with DBADM authority, create a schema called RICK with the user RICK as the owner.

```
CREATE SCHEMA RICK AUTHORIZATION RICK
```

Example 2: Create a schema that has an inventory part table and an index over the part number. Give authority on the table to user JONES.

```
CREATE SCHEMA INVENTRY

  CREATE TABLE PART (PARTNO   SMALLINT NOT NULL,
                     DESCR    VARCHAR(24),
                     QUANTITY INTEGER)

  CREATE INDEX PARTIND ON PART (PARTNO)

  GRANT ALL ON PART TO JONES
```

Example 3: Create a schema called PERS with two tables that each have a foreign key that references the other table. This is an example of a feature of the CREATE SCHEMA statement that allows such a pair of tables to be created without the use of the ALTER TABLE statement.

```
CREATE SCHEMA PERS

  CREATE TABLE ORG (DEPTNUMB  SMALLINT NOT NULL,
                    DEPTNAME VARCHAR(14),
                    MANAGER  SMALLINT,
                    DIVISION VARCHAR(10),
                    LOCATION VARCHAR(13),
                    CONSTRAINT PKEYDNO
                      PRIMARY KEY (DEPTNUMB),
                    CONSTRAINT FKEYMGR
                      FOREIGN KEY (MANAGER)
                      REFERENCES STAFF (ID) )

  CREATE TABLE STAFF (ID       SMALLINT NOT NULL,
                     NAME     VARCHAR(9),
                     DEPT     SMALLINT,
                     JOB      VARCHAR(5),
                     YEARS    SMALLINT,
                     SALARY   DECIMAL(7,2),
                     COMM     DECIMAL(7,2),
                     CONSTRAINT PKEYID
                       PRIMARY KEY (ID),
                     CONSTRAINT FKEYDNO
                       FOREIGN KEY (DEPT)
                       REFERENCES ORG (DEPTNUMB) )
```

Related reference:

- "COMMENT " on page 909
- "CREATE INDEX " on page 921
- "CREATE TABLE " on page 956
- "CREATE VIEW " on page 1034
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

CREATE SECURITY LABEL

The CREATE SECURITY LABEL statement defines a security label.

Invocation:

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
►►──CREATE SECURITY LABEL──security-label-name──────────────────────────►


   ┌─,──────────────────────────────────────────┐
   │                           ┌─,────────────┐ │
   ▼                           ▼              │ │
►────COMPONENT──component-name────string-constant─┴─┴──────────────────────►◄
```

Description:

security-label-name
: Names the security label. The name must be qualified with a security policy, and must not identify an existing security label for this security policy (SQLSTATE 42710).

COMPONENT *component-name*
: Specifies the name of a security label component. If the component is not part of the security policy *security-policy-name*, an error is returned (SQLSTATE 4274G). If a component is specified twice in the same statement, an error is returned (SQLSTATE 42713).

string-constant,**...**
: Specifies a valid element for the security component. A valid element is one that was specified when the security component was created. If the element is invalid, an error is returned (SQLSTATE 4274F).

Examples:

Example 1: Create a security label named EMPLOYEESECLABEL that is part of the DATA_ACCESS security policy, and that has the element Top Secret for the LEVEL component and the elements Research and Analysis for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Research', 'Analysis'
```

Example 2: Create a security label named EMPLOYEESECLABELREAD that only has the element Research for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELREAD
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Research'
```

Example 3: Create a security label named EMPLOYEESECLABELWRITE that only has the element Analysis for the COMPARTMENTS component.

```
CREATE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELWRITE
  COMPONENT LEVEL 'Top Secret',
  COMPONENT COMPARTMENTS 'Analysis'
```

Example 4: Create a security label named BEGINNER that is part of an existing CLASSPOLICY security policy, and that has the element Trainee for the TRUST component and the element Morning for the SECTIONS component.

```
CREATE SECURITY LABEL CLASSPOLICY.BEGINNER
  COMPONENT TRUST 'Trainee',
  COMPONENT SECTIONS 'Morning'
```

Related concepts:

- "Database authorities" on page 74

Related reference:

- "CREATE SECURITY LABEL COMPONENT " on page 952
- "CREATE SECURITY POLICY " on page 955

CREATE SECURITY LABEL COMPONENT

The CREATE SECURITY LABEL COMPONENT statement defines a component that is to be used as part of a security policy.

Invocation:

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
>>--CREATE SECURITY LABEL COMPONENT--component-name--+-| array-clause |-+------><
                                                     +-| set-clause |---+
                                                     '-| tree-clause |--'
```

array-clause:

```
                 .-,---------------.
                 V                 |
|--ARRAY--[--------string-constant---+--]-----------------------------------|
```

set-clause:

```
           ┌─,──────────────┐
           ▼                │
├──SET──{────string-constant──┴──}──────────────────────────────────────┤
```

tree-clause:

```
├──TREE──(──string-constant──ROOT──┬──────────────────────────────────────────────┬──)──────┤
                                   │ ┌──────────────────────────────────────────┐ │
                                   │ ▼                                          │ │
                                   └───,──string-constant──UNDER──string-constant─┴─┘
```

Description:

component-name
Names the security label component. This is a one-part name. The name must not identify an existing security label component at the current server (SQLSTATE 42710).

ARRAY
Specifies an ordered set of elements.

string-constant,...
One or more string constant values that make up the set of valid values for this security label component. The order in which the array elements appear is important. The first element ranks higher than the second element. The second element ranks higher than the third element and so on.

SET
Specifies an unordered set of elements.

string-constant,...
One or more string constant values that make up the set of valid values for this security label component. The order of the elements is not important.

TREE
Specifies a tree structure of node elements.

string-constant
One or more string constant values that make up the set of valid values for this security label component.

ROOT
Specifies that the *string-constant* that follows the keyword is the root node element of the tree.

UNDER
Specifies that the *string-constant* before the **UNDER** keyword is a child of the *string-constant* that follows the **UNDER** keyword. An element must be defined as either being the root element or as being the child of another element before it can be used as a parent, otherwise an error (SQLSTATE 42704) is returned.

Rules:

These rules apply to all three types of component (ARRAY, SET, and TREE):
- Element names cannot contain any of these characters:
  - Opening parenthesis - (

  – Closing parenthesis -)
  – Comma - ,
  – Colon - :
- An element name can have no more than 32 bytes (SQLSTATE 42622)
- If a security label component is a set or a tree, no more than 64 elements can be part of that component; if the component is an array, no more than 65 535 elements can be part of that component (SQLSTATE 54061)
- No element name can be used more than once in the same component (SQLSTATE 42713)

Examples:

Example 1: Create an ARRAY type security label component named LEVEL. The component has the following four elements, listed in order of decreasing rank: Top Secret, Secret, Classified, and Unclassified.

```
   CREATE SECURITY LABEL COMPONENT LEVEL
     ARRAY ['Top Secret', 'Secret', 'Classified', 'Unclassified']
```

Example 2: Create a SET type security label component named COMPARTMENTS. The component has the following three elements: Research, Analysis, and Collection.

```
   CREATE SECURITY LABEL COMPONENT COMPARTMENTS
     SET {'Collection', 'Research', 'Analysis'}
```

Example 3: Create a TREE type security label component named GROUPS. GROUPS has five elements: PROJECT, TEST, DEVELOPMENT, CURRENT, AND FIELD. The following diagram shows the relationship of these elements to one another:

```
         PROJECT
   ________|________
  |                 |
TEST          DEVELOPMENT
                ______|______
               |             |
            CURRENT        FIELD
```

```
   CREATE SECURITY LABEL COMPONENT GROUPS
     TREE (
       'PROJECT' ROOT,
       'TEST' UNDER 'PROJECT',
       'DEVELOPMENT' UNDER 'PROJECT',
       'CURRENT' UNDER 'DEVELOPMENT',
       'FIELD' UNDER 'DEVELOPMENT'
     )
```

Related concepts:
- "Database authorities" on page 74

Related reference:
- "CREATE SECURITY LABEL " on page 951
- "CREATE SECURITY POLICY " on page 955

CREATE SECURITY POLICY

The CREATE SECURITY POLICY statement defines a security policy.

Invocation:

This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
►►──CREATE SECURITY POLICY──security-policy-name──────────────────────────►


              ┌─,──────────────┐
►─COMPONENTS───▼─component-name─┴──WITH DB2LBACRULES──────────────────────►

  ┌─OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL─┐
►─┼──────────────────────────────────────────────┼───────────────────────►◄
  └─RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL─┘
```

Description:

security-policy-name
: Names the security policy. This is a one-part name. The name must not identify an existing security policy at the current server (SQLSTATE 42710).

COMPONENTS *component-name***,...**
: Identifies a security label component. The name must identify a security label component that already exists at the current server (SQLSTATE 42704). The same security component must not be specified more than once for the security policy (SQLSTATE 42713). No more than 16 security label components can be specified for a security policy (SQLSTATE 54062).

WITH DB2LBACRULES
: Indicates what rule set that will be used when comparing security labels that are part of this security policy. There is currently only one rule set: DB2LBACRULES.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL or **RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL**
: Specifies the action that is to be taken when a user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement issued against a table that is protected with this security policy. A user's security label and exemption credentials determine the user's authorization to write an explicitly provided security label. The default is OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL.

OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL
Indicates that the value of the user's security label, rather than the explicitly specified security label, is to be used for write access during an insert or update operation.

RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
Indicates that the insert or update operation will fail if the user is not authorized to write the explicitly specified security label that is provided in the INSERT or UPDATE statement (SQLSTATE 42519).

Examples:

Example 1: Create a security policy named DATA_ACCESS that uses the DB2LBACRULES rule set and has three components: LEVEL, COMPARTMENTS, and GROUPS, in that order. Assume that all three components already exist.

```
CREATE SECURITY POLICY DATA_ACCESS
  COMPONENTS LEVEL, COMPARTMENTS, GROUPS
  WITH DB2LBACRULES
```

Example 2: Create a security policy named CONTRIBUTIONS that has the components MEMBER and BADGE, which are assumed to already exist.

```
CREATE SECURITY POLICY CONTRIBUTIONS
  COMPONENTS MEMBER, BADGE
  WITH DB2LBACRULES
```

Related concepts:
- "Database authorities" on page 74

Related reference:
- "CREATE SECURITY LABEL " on page 951
- "CREATE SECURITY LABEL COMPONENT " on page 952

CREATE TABLE

The CREATE TABLE statement defines a table. The definition must include its name and the names and attributes of its columns. The definition can include other attributes of the table, such as its primary key or check constraints.

To declare a global temporary table, use the DECLARE GLOBAL TEMPORARY TABLE statement.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- CREATETAB authority on the database and USE privilege on the table space, as well as one of:

   – IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the table does not exist
   – CREATEIN privilege on the schema, if the schema name of the table refers to an existing schema
- SYSADM or DBADM authority

If a subtable is being defined, the authorization ID must be the same as the definer of the root table of the table hierarchy.

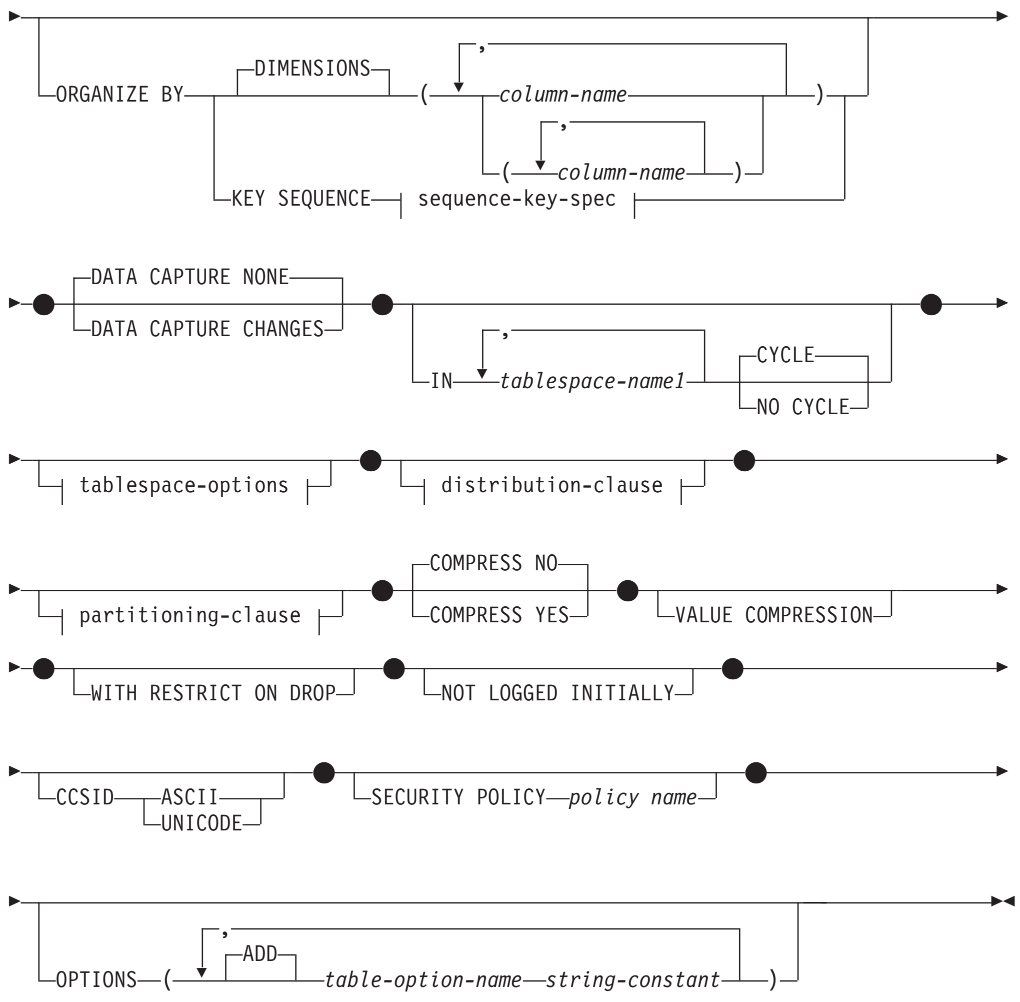
To define a foreign key, the privileges held by the authorization ID of the statement must include one of the following on the parent table:
- REFERENCES privilege on the table
- REFERENCES privilege on each column of the specified parent key
- CONTROL privilege on the table
- SYSADM or DBADM authority

To define a materialized query table (using a fullselect), the privileges held by the authorization ID of the statement must include at least one of the following on each table or view identified in the fullselect:
- SELECT privilege on the table or view, and ALTER privilege if REFRESH DEFERRED or REFRESH IMMEDIATE is specified
- CONTROL privilege on the table or view
- SYSADM or DBADM authority

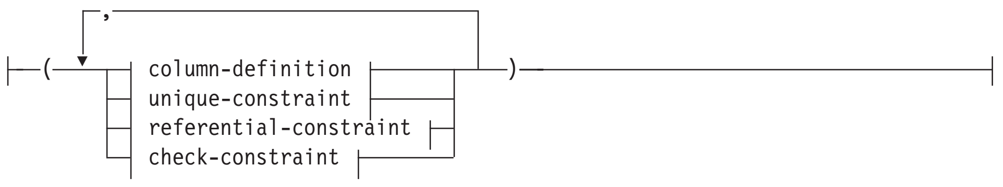
To define a staging table associated with a materialized query table, the privileges held by the authorization ID of the statement must include at least one of the following:
- CONTROL privilege or ALTER privilege on the materialized query table, and at least one of the following on each table or view identified in the fullselect of the materialized query table:
   – SELECT privilege and ALTER privilege on the table or view
   – CONTROL privilege on the table or view
- SYSADM or DBADM authority

Syntax:

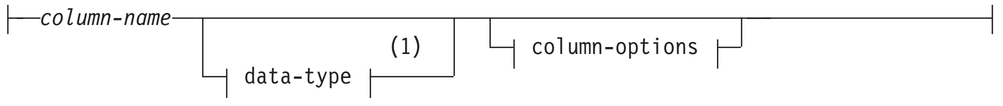
```
►►──CREATE TABLE──table-name──┬─┤ element-list ├────────────────────────────┬──●──►
                              ├─OF──type-name1──┬──────────────────────────┬┤
                              │                 └─┤ typed-table-options ├──┘│
                              ├─┤ materialized-query-definition ├───────────┤
                              ├─┤ staging-table-definition ├────────────────┤
                              └─LIKE──┬─table-name1─┬──┬──────────────────┬─┘
                                      ├─view-name───┤  └─┤ copy-options ├─┘
                                      └─nickname────┘
```



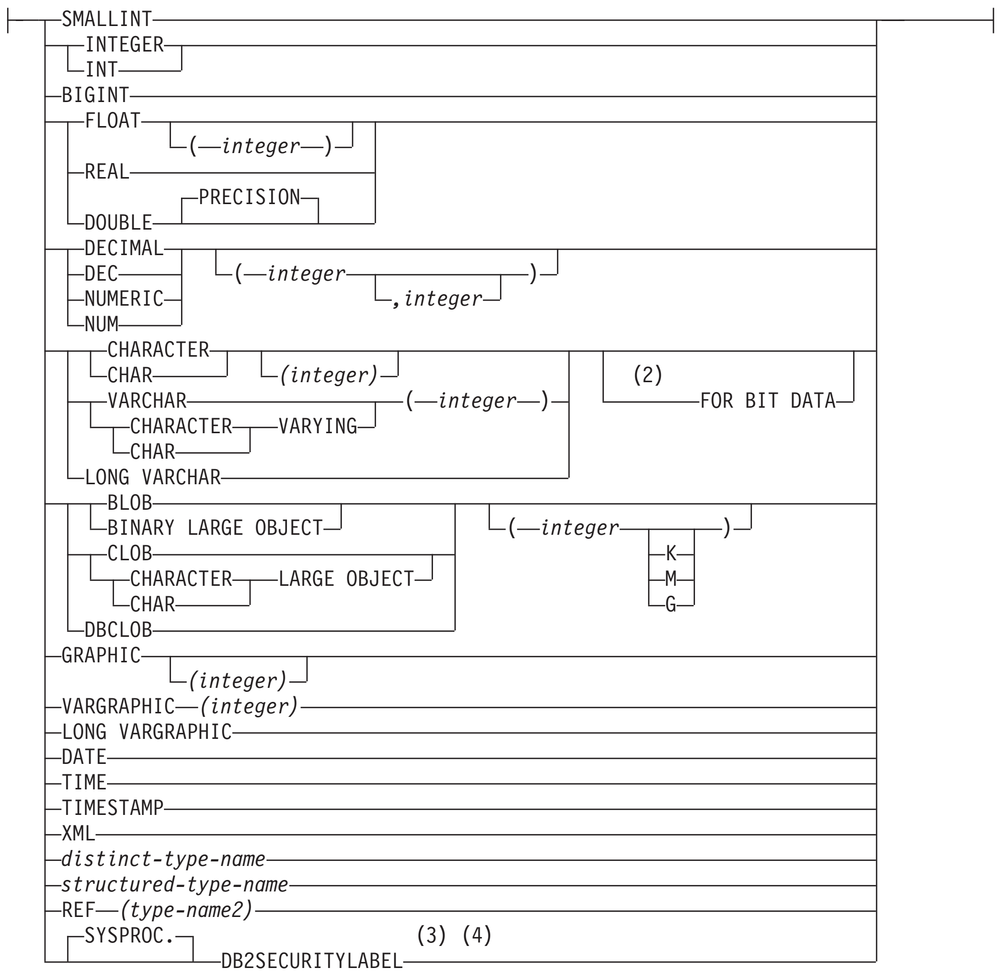
element-list:



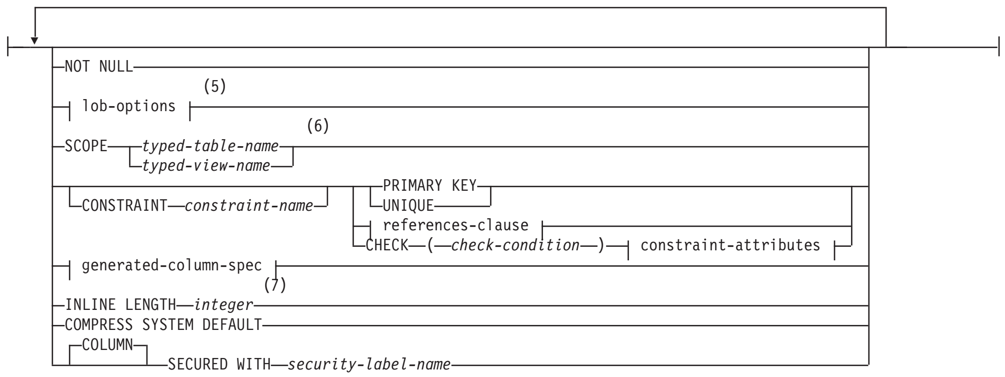
column-definition:



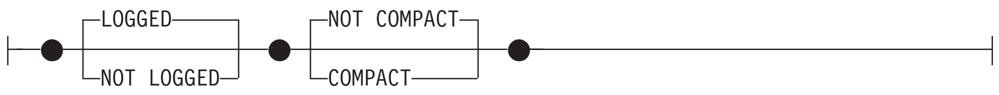
data-type:



column-options:



lob-options:



file-link-options:

```
├──●─INTEGRITY───ALL──●─READ PERMISSION──┬─FS─┬──────────────────────────────────────►
                                         └─DB─┘

►──●─WRITE PERMISSION──┬─FS──────────────────────────────────────────┬─────────────►
                       ├─BLOCKED─────────────────────────────────────┤
                       └─ADMIN──┬─────┬──REQUIRING TOKEN FOR UPDATE─┘
                                └─NOT─┘

►──●─RECOVERY──┬─NO──┬──●─ON UNLINK──┬─RESTORE─┬──●──────────────────────────────────┤
               └─YES─┘               └─DELETE──┘
```

references-clause:

```
├──REFERENCES──┬─table-name─┬──┬──────────────────────────┬──────────────────────────►
               └─nickname───┘  │      ┌─,───────────┐     │
                               │      ▼             │     │
                               └─(────column-name───┴─)─┘

►──┤ rule-clause ├──┤ constraint-attributes ├────────────────────────────────────────┤
```

rule-clause:

```
   ┌─ON DELETE NO ACTION─────────┐    ┌─ON UPDATE NO ACTION─┐
├──●──┼─────────────────────────────┼──●──┼─────────────────────┼──●─────────────────┤
   └─ON DELETE──┬─RESTRICT─┬────┘    └─ON UPDATE RESTRICT──┘
                ├─CASCADE──┤
                └─SET NULL─┘
```

constraint-attributes:

```
   ┌─ENFORCED──────┐     ┌─ENABLE QUERY OPTIMIZATION──┐
├──●──┼───────────────┼──●──┼────────────────────────────┼──●────────────────────────┤
   └─NOT ENFORCED──┘     └─DISABLE QUERY OPTIMIZATION─┘
```

generated-column-spec:

```
├──┬──┤ default-clause ├─────────────────────────────────────────────┬───────────────┤
   │                 ┌─ALWAYS─────┐                                  │
   ├─GENERATED──┼────────────┼──┬───────────────────────┬─────┤
   │                 └─BY DEFAULT─┘  └─┤ identity-options ├─┘     │
   │                 ┌─ALWAYS─┐                                    │
   └─GENERATED──┴────────┴──AS──(──generation-expression──)──┘
```

default-clause:

```
   ┌─WITH─┐
├──┴──────┴──DEFAULT──┬──────────────────────┬──────────────────────────────────────┤
                      └─┤ default-values ├──┘
```

default-values:

```
├──┬─constant─────────────────────────────────────────┬─────────────────────────┤
   ├─datetime-special-register────────────────────────┤
   ├─user-special-register────────────────────────────┤
   ├─CURRENT SCHEMA───────────────────────────────────┤
   ├─NULL─────────────────────────────────────────────┤
   └─cast-function──(──┬─constant──────────────────┬─)─┘
                       ├─datetime-special-register─┤
                       ├─user-special-register─────┤
                       └─CURRENT SCHEMA────────────┘
```

identity-options:

```
├──AS IDENTITY──┬──────────────────────────────────────────────────────────┬───────┤
                │      ┌───────────────────────────────────────────┐       │
                │      ▼  (8)                                      │       │
                └─(────────┬─START WITH──┬─1────────────────┬──┬──┴──)─┘
                           │             └─numeric-constant─┘  │
                           │               ┌─1────────────────┐ │
                           ├─INCREMENT BY──┴─numeric-constant─┴─┤
                           │ ┌─NO MINVALUE──────────────────┐    │
                           ├─┴─MINVALUE──numeric-constant─┴────┤
                           │ ┌─NO MAXVALUE──────────────────┐    │
                           ├─┴─MAXVALUE──numeric-constant─┴────┤
                           │ ┌─NO CYCLE─┐                        │
                           ├─┴─CYCLE────┴──────────────────────┤
                           │ ┌─CACHE 20───────────────────┐      │
                           ├─┼─NO CACHE───────────────────┼──────┤
                           │ └─CACHE──integer-constant────┘      │
                           │ ┌─NO ORDER─┐                        │
                           └─┴─ORDER────┴──────────────────────┘
```

unique-constraint:

```
                                                      ┌─,───────────┐
                                                      ▼             │
├──┬───────────────────────────────┬──┬─UNIQUE──────┬──(────column-name─┴──)────────┤
   └─CONSTRAINT──constraint-name─┘  └─PRIMARY KEY─┘
```

referential-constraint:

```
                                                      ┌─,───────────┐
                                                      ▼             │
├──┬───────────────────────────────┬──FOREIGN KEY──(────column-name─┴──)──────────►
   └─CONSTRAINT──constraint-name─┘

►──┤ references-clause ├──────────────────────────────────────────────────────┤
```

check-constraint:

```
├──┬───────────────────────────────┬──CHECK──(──┤ check-condition ├──)──────────►
   └─CONSTRAINT──constraint-name─┘

►──┤ constraint-attributes ├──────────────────────────────────────────────────┤
```

check-condition:

```
├──┬─search-condition─────────────┬──────────────────────────────────────────┤
   └─┤ functional-dependency ├────┘
```

functional-dependency:

```
├──┬─column-name─────────────────┬─DETERMINED BY──┬─column-name─────────────────┬──────┤
   │    ┌─,───────────┐          │                │    ┌─,───────────┐          │
   │    ▼             │          │                │    ▼             │          │
   └─(────column-name─┴──)───────┘                └─(────column-name─┴──)───────┘
```

typed-table-options:

```
├──┬───────────────────────────────┬──┬─────────────────────────────┬──────────────┤
   ├─HIERARCHY──hierarchy-name─────┤  └─┤ typed-element-list ├──────┘
   └─┤ under-clause ├──────────────┘
```

under-clause:

```
├──UNDER──supertable-name──INHERIT SELECT PRIVILEGES──────────────────────────┤
```

typed-element-list:

```
        ┌─,─────────────────────────────────────┐
        ▼                                       │
├──(──────┬─┤ OID-column-definition ├─────────┬─┴──)───────────────────────────┤
          ├─┤ with-options ├──────────────────┤
          ├─┤ unique-constraint ├─────────────┤
          └─┤ check-constraint ├──────────────┘
```

OID-column-definition:

```
├──REF IS──OID-column-name──USER GENERATED─────────────────────────────────────┤
```

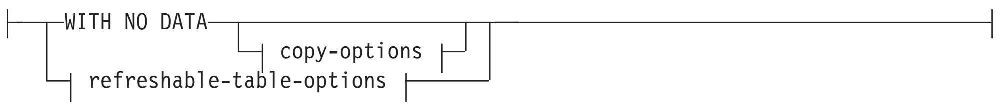
with-options:

```
├──column-name──WITH OPTIONS──┤ column-options ├───────────────────────────────┤
```

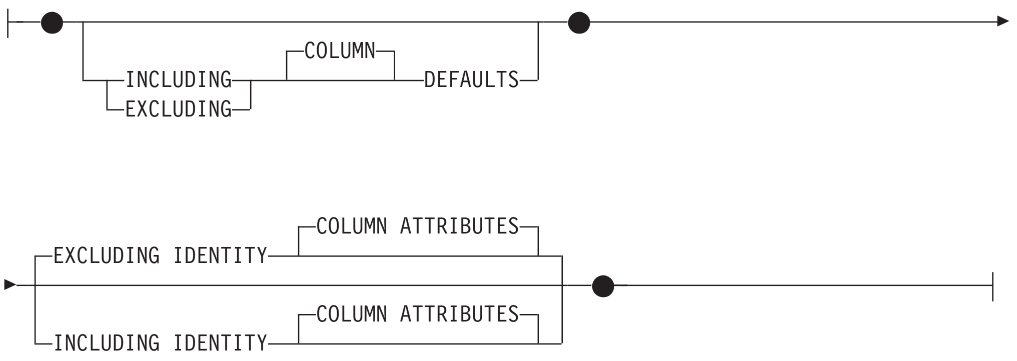
materialized-query-definition:

```
├──┬─────────────────────────────┬─AS──(──fullselect──)─────────────────────────►
   │    ┌─,───────────┐          │
   │    ▼             │          │
   └─(────column-name─┴──)───────┘

►──┤ materialized-query-table-options ├─────────────────────────────────────────┤
```

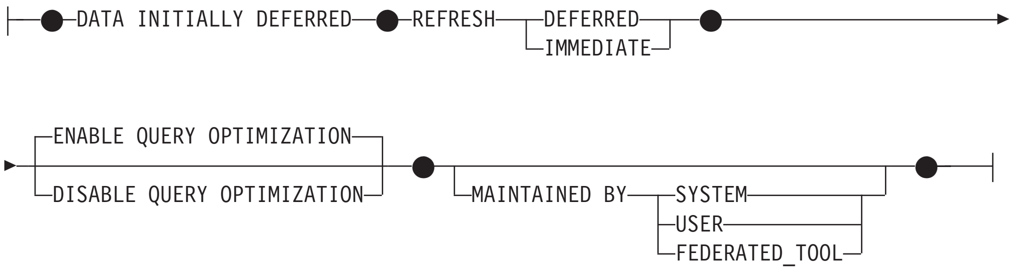
materialized-query-table-options:



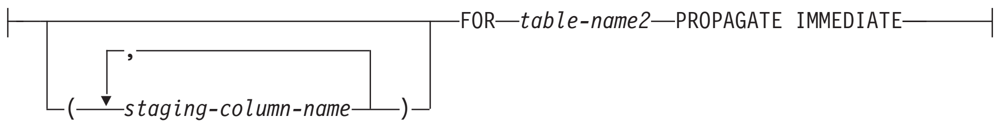
copy-options:



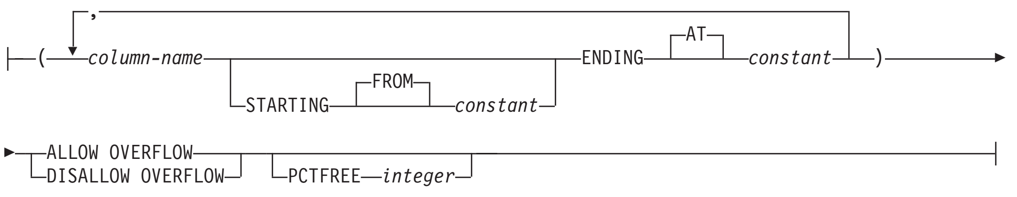
refreshable-table-options:



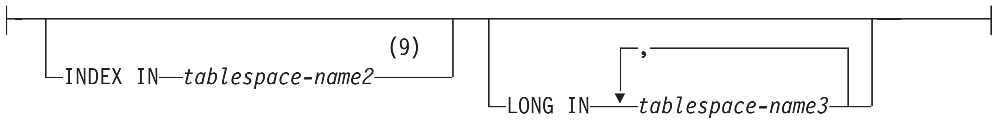
staging-table-definition:



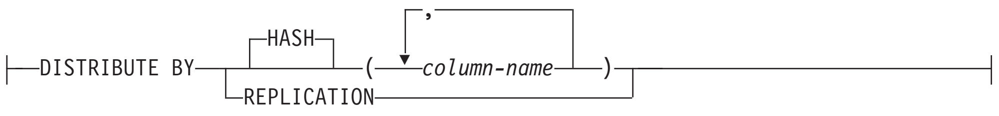
sequence-key-spec:



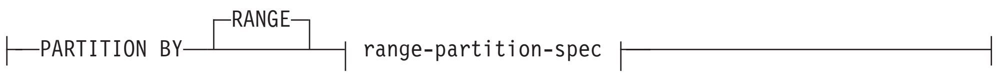
tablespace-options:



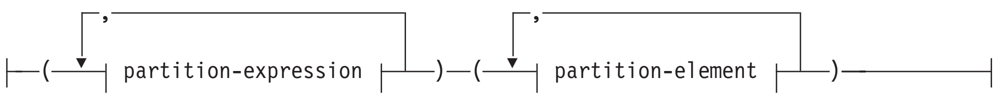
distribution-clause:



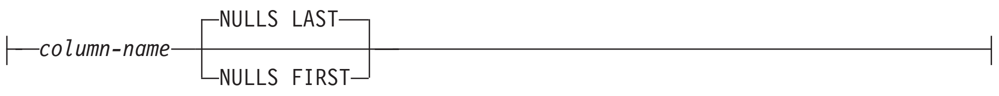
partitioning-clause:



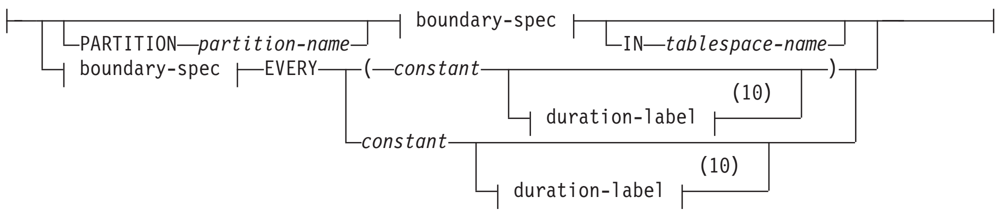
range-partition-spec:



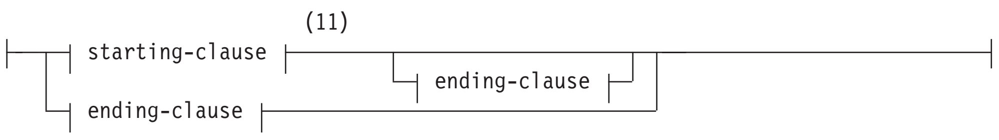
partition-expression:



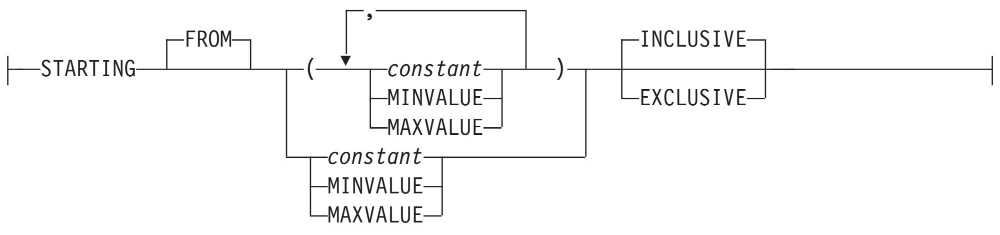
partition-element:



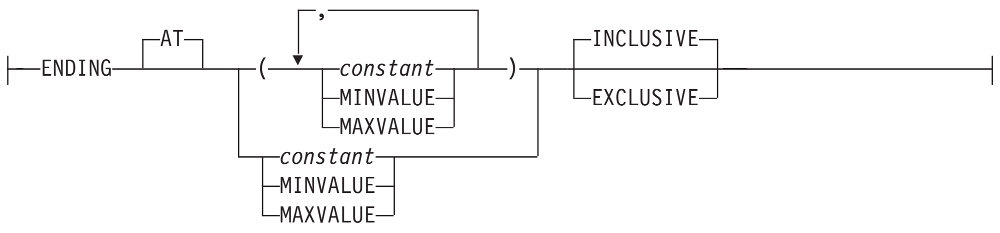
boundary-spec:



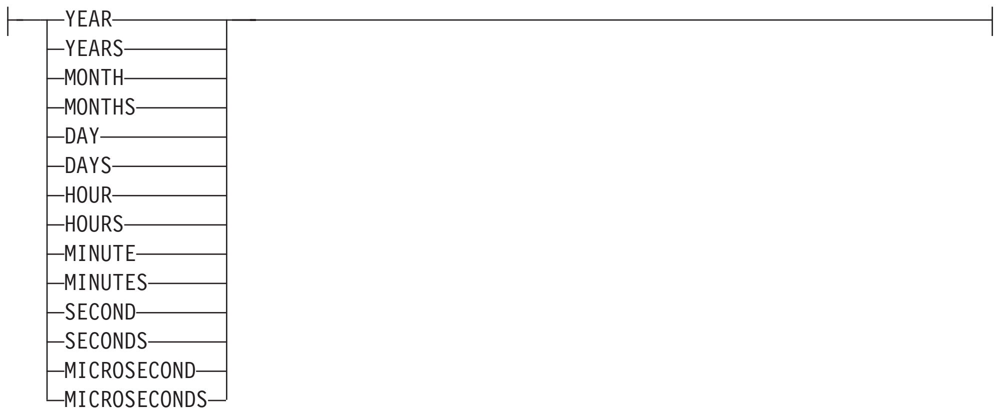
starting-clause:



ending-clause:



duration-label:



Notes:

1 If the first column-option chosen is a generated-column-spec with a generation-expression, then the data-type can be omitted. It will be determined from the resulting data type of the generation-expression.

2 The FOR BIT DATA clause can be specified in any order with the other column constraints that follow.

3 DB2SECURITYLABEL is the built-in distinct type that must be used to define the row security label column of a protected table.

4 For a column of type DB2SECURITYLABEL, NOT NULL WITH DEFAULT is implicit and cannot be explicitly specified (SQLSTATE 42842). The default value for a column of type DB2SECURITYLABEL is the session authorization ID's security label for write access.

5 The lob-options clause only applies to large object types (BLOB, CLOB and DBCLOB) and distinct types based on large object types.

6 The SCOPE clause only applies to the REF type.

7 INLINE LENGTH only applies to columns defined as structured types.

8 The same clause must not be specified more than once.

9 Specifying which table space will contain a table's indexes can be done when the table is created. If the table is a range partitioned table, the index table space can be specified with the IN clause of the CREATE INDEX statement.

10 This syntax for a partition-element is valid if there is only one partition-expression with a numeric or datetime data type.

11 The first partition-element must include a starting-clause and the last partition-element must include an ending-clause.

Description:

System-maintained materialized query tables and user-maintained materialized query tables are referred to by the common term *materialized query table*, unless there is a need to identify each one separately.

table-name
Names the table. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname, or alias described in the catalog. The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

element-list
Defines the elements of a table. This includes the definition of columns and constraints on the table.

column-definition
Defines the attributes of a column.

column-name
Names a column of the table. The name cannot be qualified, and the same name cannot be used for more than one column of the table (SQLSTATE 42711).

A table may have the following:

- A 4K page size with a maximum of 500 columns, where the byte counts of the columns must not be greater than 4 005.
- An 8K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 8 101.
- A 16K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 16 293.
- A 32K page size with a maximum of 1 012 columns, where the byte counts of the columns must not be greater than 32 677.

For more details, see "Row Size" on page 1007.

data-type
Is one of the types in the following list. Use:

SMALLINT
For a small integer.

INTEGER or **INT**
For a large integer.

BIGINT
For a big integer.

FLOAT(*integer***)**
For a single or double-precision floating-point number, depending on

the value of the *integer*. The value of the integer must be in the range 1 through 53. The values 1 through 24 indicate single precision and the values 25 through 53 indicate double-precision.

You can also specify:

REAL	For single precision floating-point.
DOUBLE	For double-precision floating-point.
DOUBLE PRECISION	For double-precision floating-point.
FLOAT	For double-precision floating-point.

DECIMAL*(precision-integer, scale-integer)* or **DEC***(precision-integer, scale-integer)*

For a decimal number. The first integer is the precision of the number; that is, the total number of digits; it may range from 1 to 31. The second integer is the scale of the number; that is, the number of digits to the right of the decimal point; it may range from 0 to the precision of the number.

If precision and scale are not specified, the default values of 5,0 are used. The words **NUMERIC** and **NUM** can be used as synonyms for **DECIMAL** and **DEC**.

CHARACTER*(integer)* or **CHAR***(integer)* or **CHARACTER** or **CHAR**

For a fixed-length character string of length *integer*, which may range from 1 to 254. If the length specification is omitted, a length of 1 character is assumed.

VARCHAR*(integer)***,** or **CHARACTER VARYING***(integer)***,** or **CHAR VARYING***(integer)*

For a varying-length character string of maximum length *integer*, which may range from 1 to 32 672.

LONG VARCHAR

For a varying-length character string with a maximum length of 32 700.

FOR BIT DATA

Specifies that the contents of the column are to be treated as bit (binary) data. During data exchange with other systems, code page conversions are not performed. Comparisons are done in binary, irrespective of the database collating sequence.

BLOB or BINARY LARGE OBJECT*(integer [K | M | G])*

For a binary large object string of the specified maximum length in bytes.

The length may be in the range of 1 byte to 2 147 483 647 bytes.

If *integer* by itself is specified, that is the maximum length.

If *integer K* (in either upper- or lowercase) is specified, the maximum length is 1 024 times *integer*. The maximum value for *integer* is 2 097 152.

If *integer M* is specified, the maximum length is 1 048 576 times *integer*. The maximum value for *integer* is 2 048.

If *integer G* is specified, the maximum length is 1 073 741 824 times *integer*. The maximum value for *integer* is 2.

If a multiple of K, M or G that calculates out to 2 147 483 648 is specified, the actual value used is 2 147 483 647 (or 2 gigabytes minus 1 byte), which is the maximum length for a LOB column.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create BLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

Any number of spaces is allowed between the integer and K, M, or G, and a space is not required. For example, all of the following are valid:

```
BLOB(50K)    BLOB(50 K)    BLOB (50   K)
```

CLOB or CHARACTER (CHAR) LARGE OBJECT*(integer [K | M | G])*
For a character large object string of the specified maximum length in bytes.

The meaning of the *integer K | M | G* is the same as for BLOB.

If the length specification is omitted, a length of 1 048 576 (1 megabyte) is assumed.

To create CLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

It is not possible to specify the FOR BIT DATA clause for CLOB columns. However, a CHAR FOR BIT DATA string can be assigned to a CLOB column, and a CHAR FOR BIT DATA string can be concatenated with a CLOB string.

DBCLOB*(integer [K | M | G])*
For a double-byte character large object string of the specified maximum length in double-byte characters.

The meaning of the *integer K | M | G* is similar to that for BLOB. The differences are that the number specified is the number of double-byte characters, and that the maximum size is 1 073 741 823 double-byte characters.

If the length specification is omitted, a length of 1 048 576 double-byte characters is assumed.

To create DBCLOB strings greater than 1 gigabyte, you must specify the NOT LOGGED option.

GRAPHIC*(integer)*
For a fixed-length graphic string of length *integer* which may range from 1 to 127. If the length specification is omitted, a length of 1 is assumed.

VARGRAPHIC*(integer)*
For a varying-length graphic string of maximum length *integer*, which may range from 1 to 16 336.

LONG VARGRAPHIC
For a varying-length graphic string with a maximum length of 16 350.

DATE
For a date.

TIME
For a time.

TIMESTAMP
For a timestamp.

XML
For an XML document. Only well-formed XML documents can be inserted into an XML column. Columns can only be of type XML if the database is defined with code set UTF-8 and the database instance has a single database partition (SQLSTATE 42997).

An XML column has the following restrictions:

- The column cannot be part of any index except an index over XML data. Therefore, it cannot be included as a column of a primary key or unique constraint (SQLSTATE 42962).
- The column cannot be a foreign key of a referential constraint (SQLSTATE 42962).
- A default value (WITH DEFAULT) cannot be specified for the column (SQLSTATE 42613). If the column is nullable, the default for the column is the null value.
- The column cannot be used in a table with a distribution key (SQLSTATE 42997).
- The column cannot be used in a range-clustered table (SQLSTATE 429BG).
- The column cannot be used in a range-partitioned table (SQLSTATE 42997).
- The column cannot be referenced in a check constraint except in a VALIDATED predicate (SQLSTATE 42621).

When a column of type XML is created, an XML path index is created on that column. A table-level XML region index is also created when the first column of type XML is created. The name of these indexes is 'SQL' followed by a character timestamp (*yymmddhhmmssxxx*). The schema name is SYSIBM.

distinct-type-name
For a user-defined type that is a distinct type. If a distinct type name is specified without a schema name, the distinct type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a distinct type, then the data type of the column is the distinct type. The length and the scale of the column are respectively the length and the scale of the source type of the distinct type.

If a column defined using a distinct type is a foreign key of a referential constraint, then the data type of the corresponding column of the primary key must have the same distinct type.

structured-type-name
For a user-defined type that is a structured type. If a structured type name is specified without a schema name, the structured type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL, and by the CURRENT PATH register for dynamic SQL).

If a column is defined using a structured type, then the static data type of the column is the structured type. The column may include values with a dynamic type that is a subtype of *structured-type-name*.

A column defined using a structured type cannot be used in a primary key, unique constraint, foreign key, index key or distribution key (SQLSTATE 42962).

If a column is defined using a structured type, and contains a reference-type attribute at any level of nesting, that reference-type attribute is unscoped. To use such an attribute in a dereference operation, it is necessary to specify a SCOPE explicitly, using a CAST specification.

REF (*type-name2***)**
For a reference to a typed table. If *type-name2* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The underlying data type of the column is based on the representation data type specified in the REF USING clause of the CREATE TYPE statement for *type-name2* or the root type of the data type hierarchy that includes *type-name2*.

SYSPROC.DB2SECURITYLABEL
This is a built-in distinct type that must be used to define the row security label column of a protected table. The underlying data type of a column of the built-in distinct type DB2SECURITYLABEL is VARCHAR(128) FOR BIT DATA. A table can have at most one column of type DB2SECURITYLABEL (SQLSTATE 428C1).

column-options
Defines additional options related to columns of the table.

NOT NULL
Prevents the column from containing null values.

If NOT NULL is not specified, the column can contain null values, and its default value is either the null value or the value provided by the WITH DEFAULT clause.

lob-options
Specifies options for LOB data types.

LOGGED
Specifies that changes made to the column are to be written to the log. The data in such columns is then recoverable with database utilities (such as RESTORE DATABASE). LOGGED is the default.

LOBs greater than 1 gigabyte cannot be logged (SQLSTATE 42993).

NOT LOGGED
Specifies that changes made to the column are not to be logged.

NOT LOGGED has no effect on a commit or rollback operation; that is, the database's consistency is maintained even if a transaction is rolled back, regardless of whether or not the LOB value is logged. The implication of not logging is that during a roll forward operation, after a backup or load operation, the LOB data will be replaced by zeros for those LOB values that would have had log records replayed during the roll forward. During crash recovery, all committed changes and changes rolled back will reflect the expected results.

COMPACT
Specifies that the values in the LOB column should take up minimal disk space (free any extra disk pages in the last group used by the LOB value), rather than leave any leftover space at the

end of the LOB storage area that might facilitate subsequent append operations. Note that storing data in this way may cause a performance penalty in any append (length-increasing) operations on the column.

NOT COMPACT
Specifies some space for insertions to assist in future changes to the LOB values in the column. This is the default.

LINKTYPE URL
This defines the type of link as a Uniform Resource Locator (URL).

NO LINK CONTROL
Specifies that there will not be any check made to determine that the file exists. Only the syntax of the URL will be checked. There is no database manager control over the file.

FILE LINK CONTROL
Specifies that a check should be made for the existence of the file. Additional options may be used to give the database manager further control over the file.

file-link-options
Additional options to define the level of database manager control of the file link.

RECOVERY
Specifies whether or not DB2 will support point in time recovery of files referenced by values in this column.

YES
DB2 will support point in time recovery of files referenced by values in this column. This value can only be specified when INTEGRITY ALL and WRITE PERMISSION BLOCKED or WRITE PERMISSION ADMIN are also specified.

NO
Specifies that point in time recovery will not be supported.

MODE DB2OPTIONS
This mode defines a set of default file link options. The defaults defined by DB2OPTIONS are:

- INTEGRITY ALL
- READ PERMISSION FS
- WRITE PERMISSION FS
- RECOVERY NO

ON UNLINK is not applicable since WRITE PERMISSION FS is used.

SCOPE
Identifies the scope of the reference type column.

A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function. Specifying the scope for a reference type column may be deferred to a subsequent ALTER TABLE statement to allow the target table to be defined, usually in the case of mutually referencing tables.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of values assigned to *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

CONSTRAINT *constraint-name*
Names the constraint. A *constraint-name* must not identify a constraint that was already specified within the same CREATE TABLE statement. (SQLSTATE 42710).

If this clause is omitted, an 18 byte long identifier that is unique among the identifiers of existing constraints defined on the table is generated by the system. (The identifier consists of ″SQL″ followed by a sequence of 15 numeric characters generated by a timestamp-based function.)

When used with a PRIMARY KEY or UNIQUE constraint, the *constraint-name* may be used as the name of an index that is created to support the constraint.

PRIMARY KEY
This provides a shorthand method of defining a primary key composed of a single column. Thus, if PRIMARY KEY is specified in the definition of column C, the effect is the same as if the PRIMARY KEY(C) clause is specified as a separate clause.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) since the primary key is inherited from the supertable.

See PRIMARY KEY within the description of the *unique-constraint* below.

UNIQUE
This provides a shorthand method of defining a unique key composed of a single column. Thus, if UNIQUE is specified in the definition of column C, the effect is the same as if the UNIQUE(C) clause is specified as a separate clause.

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3) since unique constraints are inherited from the supertable.

See UNIQUE within the description of the *unique-constraint* below.

references-clause
This provides a shorthand method of defining a foreign key composed of a single column. Thus, if a references-clause is specified in the definition of column C, the effect is the same as if that references-clause were specified as part of a FOREIGN KEY clause in which C is the only identified column.

See *references-clause* under *referential-constraint* below.

CHECK (*check-condition***)**
This provides a shorthand method of defining a check constraint that applies to a single column. See CHECK (*check-condition*) below.

generated-column-spec

default-clause
Specifies a default value for the column.

WITH
An optional keyword.

DEFAULT
Provides a default value in the event a value is not supplied on INSERT or is specified as DEFAULT on INSERT or UPDATE. If a default value is not specified following the DEFAULT keyword, the default value depends on the data type of the column as shown in "ALTER TABLE".

If a column is defined as XML, a default value cannot be specified (SQLSTATE 42613). The only possible default is NULL.

If the column is based on a column of a typed table, a specific default value must be specified when defining a default. A default value cannot be specified for the object identifier column of a typed table (SQLSTATE 42997).

If a column is defined using a distinct type, then the default value of the column is the default value of the source data type cast to the distinct type.

If a column is defined using a structured type, the *default-clause* cannot be specified (SQLSTATE 42842).

Omission of DEFAULT from a *column-definition* results in the use of the null value as the default for the column. If such a column is defined NOT NULL, then the column does not have a valid default.

default-values
Specific types of default values that can be specified are as follows.

constant
Specifies the constant as the default value for the column. The specified constant must:

- represent a value that could be assigned to the column in accordance with the rules of assignment as described in Chapter 3
- not be a floating-point constant unless the column is defined with a floating-point data type
- not have non-zero digits beyond the scale of the column data type if the constant is a decimal constant (for example, 1.234 cannot be the default for a DECIMAL(5,2) column)
- be expressed with no more than 254 bytes including the quote characters, any introducer character such as the X for a hexadecimal constant, and characters from the fully

qualified function name and parentheses when the constant is the argument of a *cast-function*.

datetime-special-register
Specifies the value of the datetime special register (CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be the data type that corresponds to the special register specified (for example, data type must be DATE when CURRENT DATE is specified).

user-special-register
Specifies the value of the user special register (CURRENT USER, SESSION_USER, SYSTEM_USER) at the time of INSERT, UPDATE, or LOAD as the default for the column. The data type of the column must be a character string with a length not less than the length attribute of a user special register. Note that USER can be specified in place of SESSION_USER and CURRENT_USER can be specified in place of CURRENT USER.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register at the time of INSERT, UPDATE, or LOAD as the default for the column. If CURRENT SCHEMA is specified, the data type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register.

NULL
Specifies NULL as the default for the column. If NOT NULL was specified, DEFAULT NULL may be specified within the same column definition but will result in an error on any attempt to set the column to the default value.

cast-function
This form of a default value can only be used with columns defined as a distinct type, BLOB or datetime (DATE, TIME or TIMESTAMP) data type. For distinct type, with the exception of distinct types based on BLOB or datetime types, the name of the function must match the name of the distinct type for the column. If qualified with a schema name, it must be the same as the schema name for the distinct type. If not qualified, the schema name from function resolution must be the same as the schema name for the distinct type. For a distinct type based on a datetime type, where the default value is a constant, a function must be used and the name of the function must match the name of the source type of the distinct type with an implicit or explicit schema name of SYSIBM. For other datetime columns, the corresponding datetime function may also be used. For a BLOB or a distinct type based on BLOB, a function must be used and the name of the function must be BLOB with an implicit or explicit schema name of SYSIBM.

constant
Specifies a constant as the argument. The constant

must conform to the rules of a constant for the source type of the distinct type or for the data type if not a distinct type. If the *cast-function* is BLOB, the constant must be a string constant.

datetime-special-register
Specifies CURRENT DATE, CURRENT TIME, or CURRENT TIMESTAMP. The source type of the distinct type of the column must be the data type that corresponds to the specified special register.

user-special-register
Specifies CURRENT USER, SESSION_USER, or SYSTEM_USER. The data type of the source type of the distinct type of the column must be a string data type with a length of at least 8 bytes. If the *cast-function* is BLOB, the length attribute must be at least 8 bytes.

CURRENT SCHEMA
Specifies the value of the CURRENT SCHEMA special register. The data type of the source type of the distinct type of the column must be a character string with a length greater than or equal to the length attribute of the CURRENT SCHEMA special register. If the cast-function is BLOB, the length attribute must be at least 8 bytes.

If the value specified is not valid, an error is returned (SQLSTATE 42894).

GENERATED
Indicates that DB2 generates values for the column. GENERATED must be specified if the column is to be considered an IDENTITY column.

ALWAYS
Specifies that DB2 will always generate a value for the column when a row is inserted into the table, or whenever the result value of the *generation-expression* changes. The result of the expression is stored in the table. GENERATED ALWAYS is the recommended value unless data propagation or unload and reload operations are being done. GENERATED ALWAYS is the required value for generated columns.

BY DEFAULT
Specifies that DB2 will generate a value for the column when a row is inserted, or updated specifying the DEFAULT clause, unless an explicit value is specified. BY DEFAULT is the recommended value when using data propagation or performing an unload and reload operation.

Although not explicitly required, a unique single-column index should be defined on the generated column to ensure uniqueness of the values.

AS IDENTITY
Specifies that the column is to be the identity column for this table. A table can only have a single IDENTITY column (SQLSTATE 428C1). The IDENTITY keyword can only be specified if the data type associated with the column is an exact numeric type with a

scale of zero, or a user-defined distinct type for which the source type is an exact numeric type with a scale of zero (SQLSTATE 42815). SMALLINT, INTEGER, BIGINT, or DECIMAL with a scale of zero, or a distinct type based on one of these types, are considered exact numeric types. By contrast, single- and double-precision floating points are considered approximate numeric data types. Reference types, even if represented by an exact numeric type, cannot be defined as identity columns.

An identity column is implicitly NOT NULL. An identity column cannot have a DEFAULT clause (SQLSTATE 42623).

START WITH *numeric-constant*
Specifies the first value for the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA). The default is MINVALUE for ascending sequences, and MAXVALUE for descending sequences.

INCREMENT BY *numeric-constant*
Specifies the interval between consecutive values of the identity column. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), and does not exceed the value of a large integer constant (SQLSTATE 42820), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA).

If this value is negative, this is a descending sequence. If this value is 0, or positive, this is an ascending sequence. The default is 1.

NO MINVALUE or **MINVALUE**
Specifies the minimum value at which a descending identity column either cycles or stops generating values, or an ascending identity column cycles to after reaching the maximum value.

NO MINVALUE
For an ascending sequence, the value is the START WITH value, or 1 if START WITH was not specified. For a descending sequence, the value is the minimum value of the data type of the column. This is the default.

MINVALUE *numeric-constant*
Specifies the numeric constant that is the minimum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be less than or equal to the maximum value (SQLSTATE 42815).

NO MAXVALUE or **MAXVALUE**
Specifies the maximum value at which an ascending identity column either cycles or stops generating values, or a descending identity column cycles to after reaching the minimum value.

NO MAXVALUE
For an ascending sequence, the value is the maximum

value of the data type of the column. For a descending sequence, the value is the START WITH value, or -1 if START WITH was not specified. This is the default.

MAXVALUE *numeric-constant*
Specifies the numeric constant that is the maximum value. This value can be any positive or negative value that could be assigned to this column (SQLSTATE 42815), without non-zero digits existing to the right of the decimal point (SQLSTATE 428FA), but the value must be greater than or equal to the minimum value (SQLSTATE 42815).

NO CYCLE or **CYCLE**
Specifies whether this identity column should continue to generate values after generating either its maximum or minimum value.

NO CYCLE
Specifies that values will not be generated for the identity column once the maximum or minimum value has been reached. This is the default.

CYCLE
Specifies that values continue to be generated for this column after the maximum or minimum value has been reached. If this option is used, after an ascending identity column reaches the maximum value, it generates its minimum value; or after a descending sequence reaches the minimum value, it generates its maximum value. The maximum and minimum values for the identity column determine the range that is used for cycling.

When CYCLE is in effect, DB2 may generate duplicate values for an identity column. Although not explicitly required, a unique, single-column index should be defined on the generated column to ensure uniqueness of the values, if unique values are desired. If a unique index exists on such an identity column and a non-unique value is generated, an error occurs (SQLSTATE 23505).

NO CACHE or **CACHE**
Specifies whether to keep some pre-allocated values in memory for faster access. If a new value is needed for the identity column, and there are none available in the cache, then the end of the new cache block must be logged. However, when a new value is needed for the identity column, and there is an unused value in the cache, then the allocation of that identity value is faster, because no logging is necessary. This is a performance and tuning option.

NO CACHE
Specifies that values for the identity column are not to be pre-allocated.

When this option is specified, the values of the identity column are not stored in the cache. In this case, every request for a new identity value results in synchronous I/O to the log.

CACHE *integer-constant*
Specifies how many values of the identity sequence are to be pre-allocated and kept in memory. When values are generated for the identity column, pre-allocating and storing values in the cache reduces synchronous I/O to the log.

If a new value is needed for the identity column and there are no unused values available in the cache, the allocation of the value involves waiting for I/O to the log. However, when a new value is needed for the identity column and there is an unused value in the cache, the allocation of that identity value can happen more quickly by avoiding the I/O to the log.

In the event of a database deactivation, either normally or due to a system failure, all cached sequence values that have not been used in committed statements are *lost*; that is, they will never be used. The value specified for the CACHE option is the maximum number of values for the identity column that could be lost in case of database deactivation. (If a database is not explicitly activated, using the ACTIVATE command or API, when the last application is disconnected from the database, an implicit deactivation occurs.)

The minimum value is 2 (SQLSTATE 42815). The default value is CACHE 20.

NO ORDER or **ORDER**
Specifies whether the identity values must be generated in order of request.

NO ORDER
Specifies that the values do not need to be generated in order of request. This is the default.

ORDER
Specifies that the values must be generated in order of request.

GENERATED ALWAYS AS (*generation-expression***)**
Specifies that the definition of the column is based on an expression. (If the expression for a GENERATED ALWAYS column includes a user-defined external function, changing the executable for the function (such that the results change for given arguments) can result in inconsistent data. This can be avoided by using the SET INTEGRITY statement to force the generation of new values.) The *generation-expression* cannot contain any of the following (SQLSTATE 42621):

- Subqueries
- XMLQUERY or XMLEXISTS expressions
- Column functions
- Dereference operations or DEREF functions
- User-defined or built-in functions that are non-deterministic
- User-defined functions using the EXTERNAL ACTION option
- User-defined functions that are not defined with NO SQL

- Host variables or parameter markers
- Special registers
- References to columns defined later in the column list
- References to other generated columns
- References to columns of type XML

The data type for the column is based on the result data type of the *generation-expression*. A CAST specification can be used to force a particular data type and to provide a scope (for a reference type only). If *data-type* is specified, values are assigned to the column according to the appropriate assignment rules. A generated column is implicitly considered nullable, unless the NOT NULL column option is used. The data type of a generated column must be one for which equality is defined. This excludes columns of type LONG VARCHAR or LONG VARGRAPHIC; LOB data types; XML; structured types; and distinct types based on any of these types (SQLSTATE 42962).

INLINE LENGTH *integer*

This option is only valid for a column defined using a structured type (SQLSTATE 42842) and indicates the maximum byte size of an instance of a structured type to store inline with the rest of the values in the row. Instances of structured types that cannot be stored inline are stored separately from the base table row, similar to the way that LOB values are handled. This takes place automatically.

The default INLINE LENGTH for a structured-type column is the inline length of its type (specified explicitly or by default in the CREATE TYPE statement). If INLINE LENGTH of the structured type is less than 292, the value 292 is used for the INLINE LENGTH of the column.

Note: The inline lengths of subtypes are not counted in the default inline length, meaning that instances of subtypes may not fit inline unless an explicit INLINE LENGTH is specified at CREATE TABLE time to account for existing and future subtypes.

The explicit INLINE LENGTH value must be at least 292 and cannot exceed 32672 (SQLSTATE 54010).

COMPRESS SYSTEM DEFAULT

Specifies that system default values are to be stored using minimal space. If the VALUE COMPRESSION clause is not specified, a warning is returned (SQLSTATE 01648), and system default values are not stored using minimal space.

Allowing system default values to be stored in this manner causes a slight performance penalty during insert and update operations on the column because of extra checking that is done.

The base data type must not be a DATE, TIME, TIMESTAMP, XML, or structured data type (SQLSTATE 42842). If the base data type is a varying-length string, this clause is ignored. String values of length 0 are automatically compressed if a table has been set with VALUE COMPRESSION.

COLUMN SECURED WITH *security-label-name*
Identifies a security label that exists for the security policy that is associated with the table. The table must have a security policy associated with it (SQLSTATE 55064).

unique-constraint
Defines a unique or primary key constraint. If the table has a distribution key, any unique or primary key must be a superset of the distribution key. A unique or primary key constraint cannot be specified for a table that is a subtable (SQLSTATE 429B3). Primary or unique keys cannot be subsets of dimensions (SQLSTATE 429BE). If the table is a root table, the constraint applies to the table and all its subtables.

CONSTRAINT *constraint-name*
Names the primary key or unique constraint.

UNIQUE (*column-name***,...)**
Defines a unique key composed of the identified columns. The identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" on page 1007. For key length limits, see "SQL limits". No LOB, LONG VARCHAR, LONG VARGRAPHIC, XML, distinct type based on one of these types, or structured type can be used as part of a unique key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the unique key cannot be the same as the set of columns in the primary key or another unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

A unique constraint cannot be specified if the table is a subtable (SQLSTATE 429B3), because unique constraints are inherited from the supertable.

The description of the table as recorded in the catalog includes the unique key and its unique index. A unique bidirectional index, which allows forward and reverse scans, will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

PRIMARY KEY (*column-name***,...)**
Defines a primary key composed of the identified columns. The clause must not be specified more than once, and the identified columns must be defined as NOT NULL. Each *column-name* must identify a column of the table, and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see "Byte Counts" on page 1007. For key length limits, see "SQL limits". No LOB, LONG VARCHAR, LONG VARGRAPHIC, XML, distinct type based on one of these types, or structured type can be used as part of a primary key, even if the length attribute of the column is small enough to fit within the index key length limit for the page size (SQLSTATE 54008).

The set of columns in the primary key cannot be the same as the set of columns in a unique key (SQLSTATE 01543). (If LANGLEVEL is SQL92E or MIA, an error is returned, SQLSTATE 42891.)

Only one primary key can be defined on a table.

A primary key cannot be specified if the table is a subtable (SQLSTATE 429B3) since the primary key is inherited from the supertable.

The description of the table as recorded in the catalog includes the primary key and its primary index. A unique bidirectional index, which allows forward and reverse scans, will automatically be created for the columns in the sequence specified with ascending order for each column. The name of the index will be the same as the *constraint-name* if this does not conflict with an existing index in the schema where the table is created. If the index name conflicts, the name will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name.

If the table has a distribution key, the columns of a *unique-constraint* must be a superset of the distribution key columns; column order is unimportant.

referential-constraint
Defines a referential constraint.

CONSTRAINT *constraint-name*
Names the referential constraint.

FOREIGN KEY (*column-name***,...)**
Defines a referential constraint with the specified *constraint-name*.

Let T1 denote the object table of the statement. The foreign key of the referential constraint is composed of the identified columns. Each name in the list of column names must identify a column of T1 and the same column must not be identified more than once.

The number of identified columns must not exceed 64, and the sum of their stored lengths must not exceed the index key length limit for the page size. For column stored lengths, see “Byte Counts” on page 1007. For key length limits, see “SQL limits”. No LOB, LONG VARCHAR, LONG VARGRAPHIC, XML, distinct type based on one of these types, or structured type column can be used as part of a foreign key (SQLSTATE 42962). There must be the same number of foreign key columns as there are in the parent key and the data types of the corresponding columns must be compatible (SQLSTATE 42830). Two column descriptions are compatible if they have compatible data types (both columns are numeric, character strings, graphic, date/time, or have the same distinct type).

references-clause
Specifies the parent table or the parent nickname, and the parent key for the referential constraint.

REFERENCES *table-name* or *nickname*
The table or nickname specified in a REFERENCES clause must identify a base table or a nickname that is described in the catalog, but must not identify a catalog table.

A referential constraint is a duplicate if its foreign key, parent key, and parent table or parent nickname are the same as the foreign key, parent key, and parent table or parent nickname of a previously specified referential constraint. Duplicate referential constraints are ignored, and a warning is returned (SQLSTATE 01543).

In the following discussion, let T2 denote the identified parent table, and let T1 denote the table being created (or altered). (T1 and T2 may be the same table).

The specified foreign key must have the same number of columns as the parent key of T2 and the description of the *n*th column of the foreign key must be comparable to the description of the *n*th column of that parent key. Datetime columns are not considered to be comparable to string columns for the purposes of this rule.

(*column-name***,...)**
The parent key of a referential constraint is composed of the identified columns. Each *column-name* must be an unqualified name that identifies a column of T2. The same column must not be identified more than once.

The list of column names must match the set of columns (in any order) of the primary key or a unique constraint that exists on T2 (SQLSTATE 42890). If a column name list is not specified, then T2 must have a primary key (SQLSTATE 42888). Omission of the column name list is an implicit specification of the columns of that primary key in the sequence originally specified.

The referential constraint specified by a FOREIGN KEY clause defines a relationship in which T2 is the parent and T1 is the dependent.

rule-clause
Specifies what action to take on dependent tables.

ON DELETE
Specifies what action is to take place on the dependent tables when a row of the parent table is deleted. There are four possible actions:

- NO ACTION (default)
- RESTRICT
- CASCADE
- SET NULL

The delete rule applies when a row of T2 is the object of a DELETE or propagated delete operation and that row has dependents in T1. Let *p* denote such a row of T2.

- If RESTRICT or NO ACTION is specified, an error occurs and no rows are deleted.
- If CASCADE is specified, the delete operation is propagated to the dependents of *p* in T1.
- If SET NULL is specified, each nullable column of the foreign key of each dependent of *p* in T1 is set to null.

SET NULL must not be specified unless some column of the foreign key allows null values. Omission of the clause is an implicit specification of ON DELETE NO ACTION.

If T1 is delete-connected to T2 through multiple paths, defining two SET NULL rules with overlapping foreign key definitions is not allowed. For example: T1 (i1, i2, i3). Rule1 with foreign key (i1, i2) and Rule2 with foreign key (i2, i3) is not allowed.

The firing order of the rules is:
1. RESTRICT
2. SET NULL OR CASCADE
3. NO ACTION

If any row in T1 is affected by two different rules, an error occurs and no rows are deleted.

A referential constraint cannot be defined if it would cause a table to be delete-connected to itself by a cycle involving two or more tables, and where one of the delete rules is RESTRICT or SET NULL (SQLSTATE 42915).

A referential constraint that would cause a table to be delete-connected to either itself or another table by multiple paths can be defined, except in the following cases (SQLSTATE 42915):
- A table must not be both a dependent table in a CASCADE relationship (self-referencing, or referencing another table), and have a self-referencing relationship in which the delete rule is RESTRICT or SET NULL.
- A key overlaps another key when at least one column in one key is the same as a column in the other key. When a table is delete-connected to another table through multiple relationships with overlapping foreign keys, those relationships must have the same delete rule, and none of the delete rules can be SET NULL.
- When a table is delete-connected to another table through multiple relationships, and at least one of those relationships is specified with a delete rule of SET NULL, the foreign key definitions of these relationships must not contain any distribution key or multidimensional clustering (MDC) key column.
- When two tables are delete-connected to the same table through CASCADE relationships, the two tables must not be delete-connected to each other if the delete rule of the last relationship in each delete-connected path is RESTRICT or SET NULL.

If any row in T1 is affected by different delete rules, the result would be the effect of all the actions specified by these rules. AFTER triggers and CHECK constraints on T1 will also see the effect of all the actions. An example of this is a row that is targeted to be set null through one delete-connected path to an ancestor table, and targeted to be deleted by a second delete-connected path to the same ancestor table. The result would be the deletion of the row. AFTER DELETE triggers on this descendant table would be activated, but AFTER UPDATE triggers would not.

In applying the above rules to referential constraints, in which either the parent table or the dependent table is a member of a typed table hierarchy, all the referential constraints that apply to any table in the respective hierarchies are taken into consideration.

ON UPDATE
Specifies what action is to take place on the dependent tables when

a row of the parent table is updated. The clause is optional. ON UPDATE NO ACTION is the default and ON UPDATE RESTRICT is the only alternative.

The difference between NO ACTION and RESTRICT is described in the "Notes" section.

check-constraint
Defines a check constraint. A *check-constraint* is a *search-condition* that must evaluate to not false or a functional dependency that is defined between columns.

CONSTRAINT *constraint-name*
Names the check constraint.

CHECK (*check-condition***)**
Defines a check constraint. The *search-condition* must be true or unknown for every row of the table.

search-condition
The *search-condition* has the following restrictions:

- A column reference must be to a column of the table being created.
- The *search-condition* cannot contain a TYPE predicate.
- The *search-condition* cannot contain any of the following (SQLSTATE 42621):
  - Subqueries
  - XMLQUERY or XMLEXISTS expressions
  - Dereference operations or DEREF functions where the scoped reference argument is other than the object identifier (OID) column
  - CAST specifications with a SCOPE clause
  - Column functions
  - Functions that are not deterministic
  - Functions defined to have an external action
  - User-defined functions defined with either CONTAINS SQL or READS SQL DATA
  - Host variables
  - Parameter markers
  - Special registers
  - References to generated columns other than the identity column
  - References to columns of type XML (except in a VALIDATED predicate)
  - An error tolerant *nested-table-expression*

functional-dependency
Defines a functional dependency between columns.

column-name **DETERMINED BY** *column-name* or **(***column-name***,...) DETERMINED BY (***column-name***,...)**
The parent set of columns contains the identified columns that immediately precede the DETERMINED BY clause. The child set of columns contains the identified columns that immediately follow the DETERMINED BY clause. All of the restrictions on the *search-condition* apply to parent set and child set columns, and only simple column references are allowed in the set of columns

(SQLSTATE 42621). The same column must not be identified more than once in the functional dependency (SQLSTATE 42709). The data type of the column must not be a LOB data type, a distinct type based on a LOB data type, an XML data type, or a structured type (SQLSTATE 42962). No column in the child set of columns can be a nullable column (SQLSTATE 42621).

If a check constraint is specified as part of a *column-definition*, a column reference can only be made to the same column. Check constraints specified as part of a table definition can have column references identifying columns previously defined in the CREATE TABLE statement. Check constraints are not checked for inconsistencies, duplicate conditions, or equivalent conditions. Therefore, contradictory or redundant check constraints can be defined, resulting in possible errors at execution time.

The *search-condition* "IS NOT NULL" can be specified; however, it is recommended that nullability be enforced directly, using the NOT NULL attribute of a column. For example, `CHECK (salary + bonus > 30000)` is accepted if salary is set to NULL, because CHECK constraints must be either satisfied or unknown, and in this case, salary is unknown. However, `CHECK (salary IS NOT NULL)` would be considered false and a violation of the constraint if salary is set to NULL.

Check constraints with *search-condition* are enforced when rows in the table are inserted or updated. A check constraint defined on a table automatically applies to all subtables of that table.

A functional dependency is not enforced by the database manager during normal operations such as insert, update, delete, or set integrity. The functional dependency might be used during query rewrite to optimize queries. Incorrect results might be returned if the integrity of a functional dependency is not maintained.

constraint-attributes
Defines attributes associated with referential integrity or check constraints.

ENFORCED or **NOT ENFORCED**
Specifies whether the constraint is enforced by the database manager during normal operations such as insert, update, or delete. The default is ENFORCED.

ENFORCED
The constraint is enforced by the database manager. ENFORCED cannot be specified for a functional dependency (SQLSTATE 42621). ENFORCED cannot be specified when a referential constraint refers to a nickname (SQLSTATE 428G7).

NOT ENFORCED
The constraint is not enforced by the database manager. This should only be specified if the table data is independently known to conform to the constraint.

ENABLE QUERY OPTIMIZATION or **DISABLE QUERY OPTIMIZATION**
Specifies whether the constraint or functional dependency can be used for query optimization under appropriate circumstances. The default is ENABLE QUERY OPTIMIZATION.

ENABLE QUERY OPTIMIZATION
The constraint is assumed to be true and can be used for query optimization.

DISABLE QUERY OPTIMIZATION
The constraint cannot be used for query optimization.

OF *type-name1*
Specifies that the columns of the table are based on the attributes of the structured type identified by *type-name1*. If *type-name1* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be an instantiable structured type (SQLSTATE 428DP) with at least one attribute (SQLSTATE 42997).

If UNDER is not specified, an object identifier column must be specified (refer to the *OID-column-definition*). This object identifier column is the first column of the table. The object ID column is followed by columns based on the attributes of *type-name1*.

HIERARCHY *hierarchy-name*
Names the hierarchy table associated with the table hierarchy. It is created at the same time as the root table of the hierarchy. The data for all subtables in the typed table hierarchy is stored in the hierarchy table. A hierarchy table cannot be directly referenced in SQL statements. A *hierarchy-name* is a *table-name*. The *hierarchy-name*, including the implicit or explicit schema name, must not identify a table, nickname, view, or alias described in the catalog. If the schema name is specified, it must be the same as the schema name of the table being created (SQLSTATE 428DQ). If this clause is omitted when defining the root table, a name is generated by the system. This name consists of the name of the table being created, followed by a unique suffix, such that the identifier is unique among the identifiers of existing tables, views, and nicknames.

UNDER *supertable-name*
Indicates that the table is a subtable of *supertable-name*. The supertable must be an existing table (SQLSTATE 42704) and the table must be defined using a structured type that is the immediate supertype of *type-name1* (SQLSTATE 428DB). The schema name of *table-name* and *supertable-name* must be the same (SQLSTATE 428DQ). The table identified by *supertable-name* must not have any existing subtable already defined using *type-name1* (SQLSTATE 42742).

The columns of the table include the object identifier column of the supertable with its type modified to be REF(*type-name1*), followed by columns based on the attributes of *type-name1* (remember that the type includes the attributes of its supertype). The attribute names cannot be the same as the OID column name (SQLSTATE 42711).

Other table options, including table space, data capture, not logged initially, and distribution key options cannot be specified. These options are inherited from the supertable (SQLSTATE 42613).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the supertable will be granted an equivalent privilege on the newly created subtable. The subtable definer is considered to be the grantor of this privilege.

typed-element-list
Defines the additional elements of a typed table. This includes the additional options for the columns, the addition of an object identifier column (root table only), and constraints on the table.

OID-column-definition
Defines the object identifier column for the typed table.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the table as the first column. An OID is required for the root table of a table hierarchy (SQLSTATE 428DX). The table must be a typed table (the OF clause must be present) that is not a subtable (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name1* (SQLSTATE 42711). The column is defined with type REF(*type-name1*), NOT NULL and a system required unique index (with a default index name) is generated. This column is referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

with-options
Defines additional options that apply to columns of a typed table.

column-name
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of a column of the table that is not also a column of a supertable (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS clause in the statement (SQLSTATE 42613).

If an option is already specified as part of the type definition (in CREATE TYPE), the options specified here override the options in CREATE TYPE.

WITH OPTIONS *column-options*
Defines options for the specified column. See *column-options* described earlier. If the table is a subtable, primary key or unique constraints cannot be specified (SQLSTATE 429B3).

materialized-query-definition
If the table definition is based on the result of a query, the table is a materialized query table based on the query.

column-name
Names the columns in the table. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the result table of the fullselect.

A list of column names must be specified if the result table of the fullselect has duplicate column names of an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

AS
Introduces the query that is used for the definition of the table and that determines the data to be included in the table.

fullselect
Defines the query on which the table is based. The resulting column definitions are the same as those for a view defined with the same query.

Every select list element must have a name (use the AS clause for expressions). The *materialized-query-definition* defines attributes of the materialized query table. The option chosen also defines the contents of the fullselect as follows.

The fullselect cannot include a *data-change-table-reference* clause (SQLSTATE 428FL).

When WITH NO DATA is specified, any valid fullselect that does not reference a typed table or a typed view can be specified.

When REFRESH DEFERRED or REFRESH IMMEDIATE is specified, the fullselect cannot include (SQLSTATE 428EC):

- References to a materialized query table, declared temporary table, or typed table in any FROM clause
- References to a view where the fullselect of the view violates any of the listed restrictions on the fullselect of the materialized query table
- Expressions that are a reference type (or distinct type based on this type)
- Functions that have any of the following attributes:
  - EXTERNAL ACTION
  - LANGUAGE SQL
  - CONTAINS SQL
  - READS SQL DATA
  - MODIFIES SQL DATA
- Functions that depend on physical characteristics (for example, DBPARTITIONNUM, HASHEDVALUE)
- Table or view references to system objects (Explain tables also should not be specified)
- Expressions that are a structured type, LOB type (or a distinct type based on a LOB type), or XML type
- References to a protected table or protected nickname

When REPLICATED is specified, the following restrictions apply:

- The GROUP BY clause is not allowed.
- The materialized query table must only reference a single table; that is, it cannot include a join.

When REFRESH IMMEDIATE is specified:

- The query must be a subselect, with the exception that UNION ALL is supported in the input table expression of a GROUP BY.
- The query cannot be recursive.
- The query cannot include:
  - References to a nickname
  - Functions that are not deterministic
  - Scalar fullselects

  - Predicates with fullselects
  - Special registers
  - SELECT DISTINCT
  - An error tolerant *nested-table-expression*
- If the FROM clause references more than one table or view, it can only define an inner join without using the explicit INNER JOIN syntax.
- When a GROUP BY clause is specified, the following considerations apply:
  - The supported column functions are SUM, COUNT, COUNT_BIG and GROUPING (without DISTINCT). The select list must contain a COUNT(*) or COUNT_BIG(*) column. If the materialized query table select list contains SUM(X), where X is a nullable argument, the materialized query table must also have COUNT(X) in its select list. These column functions cannot be part of any expressions.
  - A HAVING clause is not allowed.
  - If in a multiple partition database partition group, the distribution key must be a subset of the GROUP BY items.
- The materialized query table must not contain duplicate rows, and the following restrictions specific to this uniqueness requirement apply, depending upon whether or not a GROUP BY clause is specified.
  - When a GROUP BY clause is specified, the following uniqueness-related restrictions apply:
    - All GROUP BY items must be included in the select list.
    - When the GROUP BY contains GROUPING SETS, CUBE, or ROLLUP, the GROUP BY items and associated GROUPING column functions in the select list must form a unique key of the result set. Thus, the following restrictions must be satisfied:
      - No grouping sets can be repeated. For example, `ROLLUP(X,Y),X` is not allowed, because it is equivalent to `GROUPING SETS((X,Y),(X),(X))`.
      - If X is a nullable GROUP BY item that appears within GROUPING SETS, CUBE, or ROLLUP, then GROUPING(X) must appear in the select list.
  - When a GROUP BY clause is not specified, the following uniqueness-related restrictions apply:
    - The materialized query table's uniqueness requirement is achieved by deriving a unique key for the materialized view from one of the unique key constraints defined in each of the underlying tables. Therefore, the underlying tables must have at least one unique key constraint defined on them, and the columns of these keys must appear in the select list of the materialized query table definition.
- When MAINTAINED BY FEDERATED_TOOL is specified, only references to nicknames are allowed in a FROM clause.

When REFRESH DEFERRED is specified:
- If the materialized query table is created with the intention of providing it with an associated staging table in a later statement, the fullselect of the materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.

- If the query is recursive, the materialized query table is not used to optimize the processing of queries.

A materialized query table whose fullselect contains a GROUP BY clause is summarizing data from the tables referenced in the fullselect. Such a materialized query table is also known as a *summary table*. A summary table is a specialized type of materialized query table.

WITH NO DATA
The query is used only to define the table. The table is not populated using the results of query and the REFRESH TABLE statement cannot be used. When the CREATE TABLE statement is completed, the table is no longer considered a materialized query table.

The columns of the table are defined based on the definitions of the columns that result from the fullselect. If the fullselect references a single table in the FROM clause, select list items that are columns of that table are defined using the column name, data type, and nullability characteristic of the referenced table.

copy-options
These options specify whether or not to copy additional attributes of the source result table definition (table, view or fullselect).

INCLUDING COLUMN DEFAULTS
Column defaults for each updatable column of the source result table definition are copied. Columns that are not updatable will not have a default defined in the corresponding column of the created table.

If LIKE *table-name* is specified and *table-name* identifies a base table or declared temporary table, then INCLUDING COLUMN DEFAULTS is the default.

EXCLUDING COLUMN DEFAULTS
Columns defaults are not copied from the source result table definition.

This clause is the default, except when LIKE *table-name* is specified and *table-name* identifies a base table or declared temporary table.

INCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are copied from the source result table definition, if possible. It is possible to copy the identity column attributes, if the element of the corresponding column in the table, view, or fullselect is the name of a table column, or the name of a view column which directly or indirectly maps to the name of a base table column with the identity property. In all other cases, the columns of the new table will not get the identity property. For example:

- the select-list of the fullselect includes multiple instances of an identity column name (that is, selecting the same column more than once)
- the select list of the fullselect includes multiple identity columns (that is, it involves a join)
- the identity column is included in an expression in the select list
- the fullselect includes a set operation (union, except, or intersect).

EXCLUDING IDENTITY COLUMN ATTRIBUTES
Identity column attributes are not copied from the source result table definition.

refreshable-table-options
Define the refreshable options of the materialized query table attributes.

DATA INITIALLY DEFERRED
Data is not inserted into the table as part of the CREATE TABLE statement. A REFRESH TABLE statement specifying the *table-name* is used to insert data into the table.

REFRESH
Indicates how the data in the table is maintained.

DEFERRED
The data in the table can be refreshed at any time using the REFRESH TABLE statement. The data in the table only reflects the result of the query as a snapshot at the time the REFRESH TABLE statement is processed. System-maintained materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807). User-maintained materialized query tables defined with this attribute do allow INSERT, UPDATE, or DELETE statements.

IMMEDIATE
The changes made to the underlying tables as part of a DELETE, INSERT, or UPDATE are cascaded to the materialized query table. In this case, the content of the table, at any point-in-time, is the same as if the specified *subselect* is processed. Materialized query tables defined with this attribute do not allow INSERT, UPDATE, or DELETE statements (SQLSTATE 42807).

ENABLE QUERY OPTIMIZATION
The materialized query table can be used for query optimization under appropriate circumstances.

DISABLE QUERY OPTIMIZATION
The materialized query table will not be used for query optimization. The table can still be queried directly.

MAINTAINED BY
Specifies whether the data in the materialized query table is maintained by the system, user, or replication tool. The default is SYSTEM.

SYSTEM
Specifies that the data in the materialized query table is maintained by the system.

USER
Specifies that the data in the materialized query table is maintained by the user. The user is allowed to perform update, delete, or insert operations against user-maintained materialized query tables. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against user-maintained materialized query tables. Only a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY USER.

FEDERATED_TOOL
Specifies that the data in the materialized query table is maintained by the replication tool. The REFRESH TABLE statement, used for system-maintained materialized query tables, cannot be invoked against federated_tool-maintained materialized query tables. Only

a REFRESH DEFERRED materialized query table can be defined as MAINTAINED BY FEDERATED_TOOL.

staging-table-definition
Defines the query supported by the staging table indirectly through an associated materialized query table. The underlying tables of the materialized query table are also the underlying tables for its associated staging table. The staging table collects changes that need to be applied to the materialized query table to synchronize it with the contents of the underlying tables.

staging-column-name
Names the columns in the staging table. If a list of column names is specified, it must consist of *two* more names than there are columns in the materialized query table for which the staging table is defined. If the materialized query table is a replicated materialized query table, or the query defining the materialized query table does not contain a GROUP BY clause, the list of column names must consist of *three* more names than there are columns in the materialized query table for which the staging table is defined. Each column name must be unique and unqualified. If a list of column names is not specified, the columns of the table inherit the names of the columns of the associated materialized query table. The additional columns are named GLOBALTRANSID and GLOBALTRANSTIME, and if a third column is necessary, it is named OPERATIONTYPE.

Table 140. Extra Columns Appended in Staging Tables

Column Name	Data Type	Column Description
GLOBALTRANSID	CHAR(8) FOR BIT DATA	The global transaction ID for each propagated row
GLOBALTRANSTIME	CHAR(13) FOR BIT DATA	The timestamp of the transaction
OPERATIONTYPE	INTEGER	Operation for the propagated row, either insert, update, or delete.

A list of column names must be specified if any of the columns of the associated materialized query table duplicates any of the generated column names (SQLSTATE 42711).

FOR *table-name2*
Specifies the materialized query table that is used for the definition of the staging table. The name, including the implicit or explicit schema, must identify a materialized query table that exists at the current server defined with REFRESH DEFERRED. The fullselect of the associated materialized query table must follow the same restrictions and rules as a fullselect used to create a materialized query table with the REFRESH IMMEDIATE option.

The contents of the staging table can be used to refresh the materialized query table, by invoking the REFRESH TABLE statement, if the contents of the staging table are consistent with the associated materialized query table and the underlying source tables.

PROPAGATE IMMEDIATE
The changes made to the underlying tables as part of a delete, insert, or update operation are cascaded to the staging table in the same delete, insert, or update operation. If the staging table is not marked inconsistent,

its content, at any point-in-time, is the delta changes to the underlying table since the last refresh materialized query table.

LIKE *table-name1* **or** *view-name* **or** *nickname*

Specifies that the columns of the table have exactly the same name and description as the columns of the identified table (*table-name1*), view (*view-name*) or nickname (*nickname*). The name specified after LIKE must identify a table, view or nickname that exists in the catalog, or a declared temporary table. A typed table or typed view cannot be specified (SQLSTATE 428EC).

The use of LIKE is an implicit definition of *n* columns, where *n* is the number of columns in the identified table, view or nickname.

- If a table is identified, then the implicit definition includes the column name, data type and nullability characteristic of each of the columns of *table-name1*. If EXCLUDING COLUMN DEFAULTS is not specified, then the column default is also included.
- If a view is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each of the result columns of the fullselect defined in *view-name*.
- If a nickname is identified, then the implicit definition includes the column name, data type, and nullability characteristic of each column of *nickname*.
- If a protected table is identified in the LIKE clause, the new table inherits the same security policy and protected columns as the identified table.

Column default and identity column attributes may be included or excluded, based on the copy-attributes clauses. The implicit definition does not include any other attributes of the identified table, view or nickname. Thus the new table does not have any unique constraints, foreign key constraints, triggers, or indexes. The table is created in the table space implicitly or explicitly specified by the IN clause, and the table has any other optional clause only if the optional clause is specified.

ORGANIZE BY DIMENSIONS (*column-name***,...)**

Specifies a dimension for each column or group of columns used to cluster the table data. The use of parentheses within the dimension list specifies that a group of columns is to be treated as one dimension. The DIMENSIONS keyword is optional. A table whose definition specifies this clause is known as a multidimensional clustering (MDC) table.

A clustering block index is automatically maintained for each specified dimension, and a block index, consisting of all columns used in the clause, is maintained if none of the clustering block indexes includes them all. The set of columns used in the ORGANIZE BY clause must follow the rules for the CREATE INDEX statement.

Each column name specified in the ORGANIZE BY clause must be defined for the table (SQLSTATE 42703), and a dimension cannot occur more than once in the dimension list (SQLSTATE 42709). The table must not include any columns with data type XML (SQLSTATE 42601).

Pages of the table are arranged in blocks of equal size, which is the extent size of the table space, and all rows of each block contain the same combination of dimension values.

A table can be both a multidimensional clustering (MDC) table and a partitioned table. Columns in such a table can be used in both the *range-partition-spec* and in the MDC key. Note that table partitioning is multi-column, not multidimensional.

ORGANIZE BY KEY SEQUENCE *sequence-key-spec*
Specifies that the table is organized in ascending key sequence with a fixed size based on the specified range of key sequence values. A table organized in this way is referred to as a *range-clustered table*. Each possible key value in the defined range has a predetermined location in the physical table. The storage required for a range-clustered table must be available when the table is created, and must be sufficient to contain the number of rows in the specified range multiplied by the row size (for details on determining the space requirement, see “Row Size” on page 1007 and “Byte Counts” on page 1007).

column-name
Specifies a column of the table that is included in the unique key that determines the sequence of the range-clustered table. The data type of the column must be SMALLINT, INTEGER, or BIGINT (SQLSTATE 42611), and the columns must be defined as NOT NULL (SQLSTATE 42831). The same column must not be identified more than once in the sequence key. The number of identified columns must not exceed 64 (SQLSTATE 54008).

A unique index entry will automatically be created in the catalog for the columns in the key sequence specified with ascending order for each column. The name of the index will be SQL, followed by a character timestamp (*yymmddhhmmssxxx*), with SYSIBM as the schema name. An actual index object is not created in storage, because the table organization is ordered by this key. If a primary key or a unique constraint is defined on the same columns as the range-clustered table sequence key, this same index entry is used for the constraint.

For the key sequence specification, a check constraint exists to reflect the column constraints. If the DISALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT, and the check constraint is enforced. If the ALLOW OVERFLOW clause is specified, the name of the check constraint will be RCT_OFLOW, and the check constraint is not enforced.

STARTING FROM *constant*
Specifies the constant value at the low end of the range for *column-name*. Values less than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821). If a starting constant is not specified, the default value is 1.

ENDING AT *constant*
Specifies the constant value at the high end of the range for *column-name*. Values greater than the specified constant are only allowed if the ALLOW OVERFLOW option is specified. The value of the ending constant must be greater than the starting constant. If *column-name* is a SMALLINT or INTEGER column, the constant must be an INTEGER constant. If *column-name* is a BIGINT column, the constant must be an INTEGER or BIGINT constant (SQLSTATE 42821).

ALLOW OVERFLOW
Specifies that the range-clustered table allows rows with key values that

are outside of the defined range of values. When a range-clustered table is created to allow overflows, the rows with key values outside of the range are placed at the end of the defined range without any predetermined order. Operations involving these overflow rows are less efficient than operations on rows having key values within the defined range.

DISALLOW OVERFLOW
Specifies that the range-clustered table does not allow rows with key values that are not within the defined range of values (SQLSTATE 23513). Range-clustered tables that disallow overflows will always maintain all rows in ascending key sequence.

PCTFREE *integer*
Specifies the percentage of each page that is to be left as free space. The first row on each page is added without restriction. When additional rows are added to a page, at least *integer* percent of the page is left as free space. The value of *integer* can range from 0 to 99. A PCTFREE value of -1 in the system catalog (SYSCAT.TABLES) is interpreted as the default value. The default PCTFREE value for a table page is 0.

DATA CAPTURE
Indicates whether extra information for inter-database data replication is to be written to the log. This clause cannot be specified when creating a subtable (SQLSTATE 42613).

If the table is a typed table, then this option is not supported (SQLSTATE 428DH or 42HDR).

Data capture is incompatible with row compression (SQLSTATE 42997).

NONE
Indicates that no extra information will be logged.

CHANGES
Indicates that extra information regarding SQL changes to this table will be written to the log. This option is required if this table will be replicated and the Capture program is used to capture changes for this table from the log.

If the schema name (implicit or explicit) of the table is longer than 18 bytes, this option is not supported (SQLSTATE 42997).

IN *tablespace-name1***,...**
Identifies the table spaces in which the table will be created. The table spaces must exist, they must be in the same database partition group, and they must be all regular DMS or all large DMS or all SMS table spaces (SQLSTATE 42838) on which the authorization ID of the statement holds the USE privilege.

A maximum of one IN clause is allowed at the table level. All data table spaces used by a table must have the same page size and extent size. If they do not all have the same prefetch size, a warning is returned. If all table spaces have AUTOMATIC prefetch size, no warning is returned.

If only one table space is specified, all table parts are stored in this table space. This clause cannot be specified when creating a subtable (SQLSTATE 42613), because the table space is inherited from the root table of the table hierarchy. If this clause is not specified, a table space for the table is determined as follows:

```
IF table space IBMDEFAULTGROUP (over which the user
  has USE privilege) exists with sufficient page size
    THEN choose it
ELSE IF a table space (over which the user has USE privilege)
```

```
  exists with sufficient page size (see below when
  multiple table spaces qualify)
    THEN choose it
ELSE return an error (SQLSTATE 42727)
```

If more than one table space is identified by the ELSE IF condition, choose the table space with the smallest sufficient page size. If more than one table space qualifies, choose the table space in the following order of preference, depending on to whom the USE privilege was granted:

1. The authorization ID
2. A group to which the authorization ID belongs
3. PUBLIC

If more than one table space still qualifies, the final choice is made by the database manager.

Table space determination can change if:
- Table spaces are dropped or created
- USE privileges are granted or revoked

Partitioned tables can have their partitions spread across multiple table spaces. When multiple table spaces are specified, all of the table spaces must exist, and they must all be either SMS or regular DMS or large DMS table spaces (SQLSTATE 42838). The authorization ID of the statement must hold the USE privilege on all of the specified table spaces.

The sufficient page size of a table is determined by either the byte count of the row or the number of columns. For more information, see "Row Size" on page 1007.

When a table is placed in a large table space:
- The table can be larger than a table in a regular table space. For details on table and table space limits, see "SQL limits".
- The table can support more than 255 rows per data page, which can improve space utilization on data pages.
- Indexes that are defined on the table will require an additional 2 bytes per row entry, compared to indexes defined on a table that resides in a regular table space.

CYCLE or **NO CYCLE**
Specifies whether or not the number of data partitions with no explicit table space can exceed the number of specified data partitions.

CYCLE
Specifies that if the number of data partitions with no explicit table space exceeds the number of specified data partitions, the table spaces are assigned to data partitions in a round-robin fashion.

NO CYCLE
Specifies that the number of data partitions with no explicit table space must not exceed the number of specified data partitions (SQLSTATE 428G1). This option prevents the round-robin assignment of table spaces to data partitions.

tablespace-options
Specifies the table space in which indexes or long column values are to be stored. For details on types of table spaces, see "CREATE TABLESPACE".

INDEX IN *tablespace-name2*

Identifies the table space in which any indexes on the table are to be created. The specified table space must exist; it must be a DMS table space if the table has data in DMS table spaces, or an SMS table space if the partitioned table has data in SMS table spaces; it must be a table space on which the authorization ID of the statement holds the USE privilege; and it must be in the same database partition group as *tablespace-name1* (SQLSTATE 42838).

Specifying which table space will contain indexes can be done when a table is created or, in the case of partitioned tables, it can be done by specifying the IN clause of the CREATE INDEX statement. Checking for the USE privilege on the table space is done at table creation time, not when an index is created later.

LONG IN *tablespace-name3*

Identifies the table spaces in which the values of any long columns are to be stored. Long columns include those with type LONG VARCHAR or LONG VARGRAPHIC, LOB data types, XML type, distinct types with any of these as source types, or any columns defined with user-defined structured types whose values cannot be stored inline. This option is allowed only if the IN clause identifies a DMS table space.

The specified table space must exist; it must be a large DMS table space on which the authorization ID of the statement holds the USE privilege; and it must be in the same database partition group as *tablespace-name1* (SQLSTATE 42838).

Specifying which table space will contain long, LOB, or XML columns can only be done when a table is created. Checking for the USE privilege is done at table creation time, not when a long or LOB column is added later.

distribution-clause

Specifies the database partitioning or the way the data is distributed across multiple database partitions.

DISTRIBUTE BY HASH (*column-name***,...)**

Specifies the use of the default hashing function on the specified columns, called a *distribution key*, as the distribution method across database partitions. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column whose data type is LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, XML, distinct type based on any of these types, or structured type can be used as part of a distribution key (SQLSTATE 42962). A distribution key cannot be specified for a table that is a subtable (SQLSTATE 42613), because the distribution key is inherited from the root table in the table hierarchy or a table with a column of data type XML (SQLSTATE 42997). If this clause is not specified, and the table resides in a multiple partition database partition group with multiple database partitions, the distribution key is defined as follows:

- If the table is a typed table, the object identifier column is the distribution key.
- If a primary key is defined, the first column of the primary key is the distribution key.

- Otherwise, the first column whose data type is valid for a distribution key becomes the distribution key.

The columns of the distribution key must be a subset of the columns that make up any unique constraints.

If none of the columns satisfies the requirements for a default distribution key, the table is created without one. Such tables are allowed only in table spaces that are defined on single-partition database partition groups.

For tables in table spaces that are defined on single-partition database partition groups, any collection of columns with data types that are valid for a distribution key can be used to define the distribution key. If you do not specify this clause, no distribution key is created.

For restrictions related to the distribution key, see "Rules" on page 1003.

DISTRIBUTE BY REPLICATION
Specifies that the data stored in the table is physically replicated on each database partition of the database partition group for the table spaces in which the table is defined. This means that a copy of all of the data in the table exists on each database partition. This option can only be specified for a materialized query table (SQLSTATE 42997).

partitioning-clause
Specifies how the data is partitioned within a database partition.

PARTITION BY RANGE *range-partition-spec*
Specifies the range partitioning scheme for the table.

partition-expression
Specifies the key data over which the range is defined to determine the target data partition of the data.

column-name
Identifies a column of the data partitioning key. The *column-name* must be an unqualified name that identifies a column of the table (SQLSTATE 42703). The same column must not be identified more than once (SQLSTATE 42709). No column with a data type that is a LONG VARCHAR, LONG VARGRAPHIC, BLOB, CLOB, DBCLOB, distinct type based on any of these types, or structured type can be used as part of a data partitioning key (SQLSTATE 42962). The number of identified columns must not exceed 16 (SQLSTATE 54008).

NULLS LAST
Indicates that null values compare high.

NULLS FIRST
Indicates that null values compare low.

partition-element
Specifies ranges for a data partitioning key and the table space where rows of the table in the range will be stored.

PARTITION *partition-name*
Names the data partition. The name must not be the same as any other data partition for the table (SQLSTATE 42710). If this clause is not specified, the name will be 'PART' followed by the character form of an integer value to make the name unique for the table.

boundary-spec
Specifies the boundaries of a range partition. The lowest range

partition must include a starting-clause, and the highest range partition must include an ending-clause (SQLSTATE 56016). Range partitions between the lowest and the highest can include either a starting-clause, ending-clause, or both clauses. If only the ending-clause is specified, the previous range partition must also have included an ending-clause (SQLSTATE 56016).

starting-clause
Specifies the low end of the range for a data partition. There must be at least one starting value specified and no more values than the number of columns in the data partitioning key (SQLSTATE 53038). If there are fewer values specified than the number of columns, the remaining values are implicitly MINVALUE.

STARTING FROM
Introduces the *starting-clause*.

constant
Specifies a constant value with a data type that is assignable to the data type of the *column-name* to which it corresponds (SQLSTATE 53045). The value must not be in the range of any other boundary-spec for the table (SQLSTATE 56016).

MINVALUE
Specifies a value that is lower than the lowest possible value for the data type of the *column-name* to which it corresponds.

MAXVALUE
Specifies a value that is greater than the greatest possible value for the data type of the *column-name* to which it corresponds.

INCLUSIVE
Indicates that the specified range values are to be included in the data partition.

EXCLUSIVE
Indicates that the specified *constant* values are to be excluded from the data partition. This specification is ignored when MINVALUE or MAXVALUE is specified.

ending-clause
Specifies the high end of the range for a data partition. There must be at least one starting value specified and no more values than the number of columns in the data partitioning key (SQLSTATE 53038). If there are fewer values specified than the number of columns, the remaining values are implicitly MAXVALUE.

ENDING AT
Introduces the *ending-clause*.

constant
Specifies a constant value with a data type that is assignable to the data type of the *column-name* to which it corresponds (SQLSTATE 53045). The value must not be in the range of any other boundary-spec for the table (SQLSTATE 56016).

MINVALUE
Specifies a value that is lower than the lowest possible value for the data type of the *column-name* to which it corresponds.

MAXVALUE
Specifies a value that is greater than the greatest possible value for the data type of the *column-name* to which it corresponds.

INCLUSIVE
Indicates that the specified range values are to be included in the data partition.

EXCLUSIVE
Indicates that the specified *constant* values are to be excluded from the data partition. This specification is ignored when MINVALUE or MAXVALUE is specified.

IN *tablespace-name*
Specifies the table space where the data partition is to be stored. The named table space must have the same page size, be in the same database partition group, and manage space in the same way as the other table spaces of the partitioned table (SQLSTATE 42838). If this clause is not specified, a table space is assigned by default in a round-robin fashion from the list of table spaces specified for the table. If a table space was not specified for large objects using the LONG IN clause, large objects are placed in the same table space as are the rest of the rows for the data partition. For partitioned tables, the LONG IN clause can be used to provide a list of table spaces. This list is used in round robin-fashion to place large objects for each data partition.

If the INDEX IN clause is not specified on the CREATE TABLE or the CREATE INDEX statement, the index is placed in the same table space as the first visible or attached partition of the table.

EVERY (*constant***)**
Specifies the width of each data partition range when using the automatically generated form of the syntax. Data partitions will be created starting at the STARTING FROM value and containing this number of values in the range. This form of the syntax is only supported for tables that are partitioned by a single numeric or datetime column (SQLSTATE 53038).

If the partitioning key column is a numeric type, the starting value of the first partition is the value specified in the starting-clause. The ending value for the first and all other partitions is calculated by adding the starting value of the partition to the increment value specified as *constant* in the EVERY clause. The starting value for all other partitions is calculated by taking the starting value for the previous partition and adding the increment value specified as *constant* in the EVERY clause.

If the partitioning key column is a DATE or a TIMESTAMP, the starting value of the first partition is the value specified in the starting-clause. The ending value for the first and all other partitions is calculated by adding the starting value of the partition to the increment value specified as a labeled duration in the EVERY clause. The starting value for all other partitions is calculated by taking the starting value for the previous partition and adding the increment value specified as a labeled duration in the EVERY clause.

For a numeric column, the EVERY value must be a positive numeric constant, and for a datetime column, the EVERY value must be a labeled duration (SQLSTATE 53045).

COMPRESS
Specifies whether or not data compression applies to the rows of the table.

YES
Specifies that data row compression is to be enabled. The data rows in the table are not subject to compression until a compression dictionary has been created with the non-inplace reorg utility or the inspect rowcompestimate utility.

NO
Specifies that data row compression is to be disabled.

VALUE COMPRESSION
This determines the row format that is to be used. Each data type has a different byte count depending on the row format that is used. For more information, see “Byte Counts” in “CREATE TABLE”. If the table is a typed table, this option is only supported on the root table of the typed table hierarchy (SQLSTATE 428DR).

The NULL value is stored using three bytes. This is the same or less space than when VALUE COMPRESSION is not active for columns of all data types, with the exception of CHAR(1). Whether or not a column is defined as nullable has no affect on the row size calculation. The zero-length data values for columns whose data type is VARCHAR, VARGRAPHIC, LONG VARCHAR, LONG VARGRAPHIC, CLOB, DBCLOB, BLOB, or XML are to be stored using two bytes only, which is less than the storage required when VALUE COMPRESSION is not active. When a column is defined using the COMPRESS SYSTEM DEFAULT option, this also allows the system default value for the column to be stored using three bytes of total storage. The row format that is used to support this determines the byte counts for each data type, and tends to cause data fragmentation when updating to or from NULL, a zero-length value, or the system default value.

WITH RESTRICT ON DROP
Indicates that the table cannot be dropped, and that the table space that contains the table cannot be dropped.

NOT LOGGED INITIALLY
Any changes made to the table by an Insert, Delete, Update, Create Index, Drop Index, or Alter Table operation in the same unit of work in which the table is created are not logged. For other considerations when using this option, see the “Notes” section of this statement.

All catalog changes and storage related information are logged, as are all operations that are done on the table in subsequent units of work.

Note: If non-logged activity occurs against a table that has the NOT LOGGED INITIALLY attribute activated, and if a statement fails (causing a rollback), or a ROLLBACK TO SAVEPOINT is executed, the entire unit of work is rolled back (SQL1476N). Furthermore, the table for which the NOT LOGGED INITIALLY attribute was activated is marked inaccessible after the rollback has occurred, and can only be dropped.

Therefore, the opportunity for errors within the unit of work in which the NOT LOGGED INITIALLY attribute is activated should be minimized.

CCSID

Specifies the encoding scheme for string data stored in the table. If the CCSID clause is not specified, the default is CCSID UNICODE for Unicode databases, and CCSID ASCII for all other databases.

ASCII

Specifies that string data is encoded in the database code page. If the database is a Unicode database, CCSID ASCII cannot be specified (SQLSTATE 56031).

UNICODE

Specifies that string data is encoded in Unicode. If the database is a Unicode database, character data is in UTF-8, and graphic data is in UCS-2. If the database is not a Unicode database, character data is in UTF-8.

If the database is not a Unicode database, tables can be created with CCSID UNICODE, but the following rules apply:

- The alternate collating sequence must be specified in the database configuration before creating the table (SQLSTATE 56031). CCSID UNICODE tables collate with the alternate collating sequence specified in the database configuration.
- Tables or table functions created with CCSID ASCII, and tables or table functions created with CCSID UNICODE, cannot both be used in a single SQL statement (SQLSTATE 53090). This applies to tables and table functions referenced directly in the statement, as well as to tables and table functions referenced indirectly (such as, for example, through referential integrity constraints, triggers, materialized query tables, and tables in the body of views).
- Tables created with CCSID UNICODE cannot be referenced in SQL functions or SQL methods (SQLSTATE 560C0).
- An SQL statement that references a table created with CCSID UNICODE cannot invoke an SQL function or SQL method (SQLSTATE 53090).
- Graphic types and user-defined types cannot be used in CCSID UNICODE tables (SQLSTATE 560C1).
- Tables cannot have both the CCSID UNICODE clause and the DATA CAPTURE CHANGES clause specified (SQLSTATE 42613).
- The Explain tables cannot be created with CCSID UNICODE (SQLSTATE 55002).
- Declared global temporary tables cannot be created with CCSID UNICODE (SQLSTATE 56031).
- CCSID UNICODE tables cannot be created in a CREATE SCHEMA statement (SQLSTATE 53090).
- The exception table for a load operation must have the same CCSID as the target table for the operation (SQLSTATE 428A5).
- The exception table for a SET INTEGRITY statement must have the same CCSID as the target table for the statement (SQLSTATE 53090).
- The target table for event monitor data must not be declared as CCSID UNICODE (SQLSTATE 55049).
- Statements that reference a CCSID UNICODE table can only be invoked from a DB2 Version 8.1 or later client (SQLSTATE 42997).

- SQL statements are always interpreted in the database code page. In particular, this means that every character in literals, hex literals, and delimited identifiers must have a representation in the database code page; otherwise, the character will be replaced with the substitution character.

Host variables in the application are always in the application code page, regardless of the CCSID of any tables in the SQL statements that are invoked. DB2 will perform code page conversions as necessary to convert data between the application code page and the section code page. The registry variable DB2CODEPAGE can be set at the client to change the application code page.

SECURITY POLICY
Names the security policy to be associated with the table.

policy-name
Identifies a security policy that already exists at the current server (SQLSTATE 42704).

OPTIONS (ADD *table-option-name string-constant*, **...)**
Table options are used to identify the remote base table. The *table-option-name* is the name of the option. The *string-constant* specifies the setting for the table option. The *string-constant* must be enclosed in single quotation marks.

The remote server (the server name that was specified in the CREATE SERVER statement) must be specified in the OPTIONS clause. The OPTIONS clause can also be used to override the schema or the unqualified name of the remote base table that is being created.

It is recommended that a schema name be specified. If a remote schema name is not specified, the qualifier for the table name is used. If the table name has no qualifier, the authorization ID of the statement is used.

If an unqualified name for the remote base table is not specified, *table-name* is used.

Rules:

- The sum of the byte counts of the columns, including the inline lengths of all structured type columns, must not be greater than the row size limit that is based on the page size of the table space (SQLSTATE 54010). For more information, see “Byte Counts” on page 1007. For typed tables, the byte count is applied to the columns of the root table of the table hierarchy, and every additional column introduced by every subtable in the table hierarchy (additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable). There is also an additional 4 bytes of overhead to identify the subtable to which each row belongs.
- The number of columns in a table cannot exceed 1 012 (SQLSTATE 54011). For typed tables, the total number of attributes of the types of all of the subtables in the table hierarchy cannot exceed 1010.
- An object identifier column of a typed table cannot be updated (SQLSTATE 42808).
- Any unique or primary key constraint defined on the table must be a superset of the distribution key (SQLSTATE 42997).
- The following rules only apply to multiple database partition databases.
  - Tables composed only of columns with types LOB, LONG VARCHAR, LONG VARGRAPHIC, XML, a distinct type based on one of these types, or a

  structured type can only be created in table spaces that are defined on single-partition database partition groups.
  - The distribution key definition of a table in a table space that is defined on a multiple partition database partition group cannot be altered.
  - The distribution key column of a typed table must be the OID column.
  - Columns of type XML cannot be used.
  - Partitioned staging tables are not supported.
- The following restrictions apply to range-clustered tables:
  - A range-clustered table cannot be specified in a database with multiple database partitions (SQLSTATE 42997).
  - A clustering index cannot be created.
  - Altering the table to add a column is not supported.
  - Altering the table to change the data type of a column is not supported.
  - Altering the table to change PCTFREE is not supported.
  - Altering the table to set APPEND ON is not suported.
  - DETAILED statistics are not available.
  - The load utility cannot be used to populate the table.
  - Columns cannot be of type XML.
- A table is not protected unless it has a security policy associated with it and it includes either a column of type DB2SECURITYLABEL or a column defined with the SECURED WITH clause. The former indicates that the table is a protected table with **row level granularity** and the latter indicates that the table a protected table with **column level granularity**.
- Declaring a column of type DB2SECURITYLABEL fails if the table does not have a security policy associated with it (SQLSTATE 55064).
- A security policy cannot be added to a typed table (SQLSTATE 428DH), materialized query table, or staging table (SQLSTATE 428FG).
- An error tolerant *nested-table-expression* cannot be specified in the fullselect of a *materialized-query-definition* (SQLSTATE 428GG).

Notes:
- Creating a table with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- If a foreign key is specified:
  - All packages with a delete usage on the parent table are invalidated.
  - All packages with an update usage on at least one column in the parent key are invalidated.
- Creating a subtable causes invalidation of all packages that depend on any table in table hierarchy.
- VARCHAR and VARGRAPHIC columns that are greater than 4 000 and 2 000 respectively should not be used as input parameters in functions in SYSFUN schema. Errors will occur when the function is invoked with an argument value that exceeds these lengths (SQLSTATE 22001).
- The use of NO ACTION or RESTRICT as delete or update rules for referential constraints determines when the constraint is enforced. A delete or update rule of RESTRICT is enforced *before* all other constraints, including those referential constraints with modifying rules such as CASCADE or SET NULL. A delete or update rule of NO ACTION is enforced *after* other referential constraints. One

example where different behavior is evident involves the deletion of rows from a view that is defined as a UNION ALL of related tables.

```
Table T1 is a parent of table T3; delete rule as noted below.
Table T2 is a parent of table T3; delete rule CASCADE.

CREATE VIEW V1 AS SELECT * FROM T1 UNION ALL SELECT * FROM T2

DELETE FROM V1
```

If table T1 is a parent of table T3 with a delete rule of RESTRICT, a restrict violation will be raised (SQLSTATE 23001) if there are any child rows for parent keys of T1 in T3.

If table T1 is a parent of table T3 with a delete rule of NO ACTION, the child rows may be deleted by the delete rule of CASCADE when deleting rows from T2 before the NO ACTION delete rule is enforced for the deletes from T1. If deletes from T2 did not result in deleting all child rows for parent keys of T1 in T3, then a constraint violation will be raised (SQLSTATE 23504).

Note that the SQLSTATE returned is different depending on whether the delete or update rule is RESTRICT or NO ACTION.

- For tables in table spaces defined on multiple partition database partition groups, table collocation should be considered when choosing the distribution keys. Following is a list of items to consider:
  - The tables must be in the same database partition group for collocation. The table spaces may be different, but must be defined in the same database partition group.
  - The distribution keys of the tables must have the same number of columns, and the corresponding key columns must be database partition-compatible for collocation.
  - The choice of distribution key also has an impact on performance of joins. If a table is frequently joined with another table, you should consider the joining column(s) as a distribution key for both tables.
- The NOT LOGGED INITIALLY option is useful for situations where a large result set needs to be created with data from an alternate source (another table or a file) and recovery of the table is not necessary. Using this option will save the overhead of logging the data. The following considerations apply when this option is specified:
  - When the unit of work is committed, all changes that were made to the table during the unit of work are flushed to disk.
  - When you run the rollforward utility and it encounters a log record that indicates that a table in the database was either populated by the Load utility or created with the NOT LOGGED INITIALLY option, the table will be marked as unavailable. The table will be dropped by the rollforward utility if it later encounters a DROP TABLE log. Otherwise, after the database is recovered, an error will be issued if any attempt is made to access the table (SQLSTATE 55019). The only operation permitted is to drop the table.
  - Once such a table is backed up as part of a database or table space back up, recovery of the table becomes possible.
- A REFRESH DEFERRED system-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION is set such that it includes system-maintained materialized query tables. A REFRESH DEFERRED user-maintained materialized query table defined with ENABLE QUERY OPTIMIZATION can be used to optimize the processing of queries if CURRENT REFRESH AGE is set to ANY and CURRENT MAINTAINED TABLE TYPES

FOR OPTIMIZATION is set such that it includes user-maintained materialized query tables. A REFRESH IMMEDIATE materialized query table defined with ENABLE QUERY OPTIMIZATION is always considered for optimization. For this optimization to be able to use a REFRESH DEFERRED or a REFRESH IMMEDIATE materialized query table, the fullselect must conform to certain rules in addition to those already described. The fullselect must:

  – be a subselect with a GROUP BY clause or a subselect with a single table reference
  – not include DISTINCT anywhere in the select list
  – not include any special registers
  – not include functions that are not deterministic.

  If the query specified when creating a materialized query table does not conform to these rules, a warning is returned (SQLSTATE 01633).

- If a materialized query table is defined with REFRESH IMMEDIATE, or a staging table is defined with PROPAGATE IMMEDIATE, it is possible for an error to occur when attempting to apply the change resulting from an insert, update, or delete operation on an underlying table. The error will cause the failure of the insert, update, or delete operation on the underlying table.
- Materialized query tables or staging tables cannot be used as exception tables when constraints are checked in bulk, such as during load operations or during execution of the SET INTEGRITY statement.
- Certain operations cannot be performed on a table that is referenced by a materialized query table defined with REFRESH IMMEDIATE, or defined with REFRESH DEFERRED with an associated staging table:
  – IMPORT REPLACE cannot be used.
  – ALTER TABLE NOT LOGGED INITIALLY WITH EMPTY TABLE cannot be done.
- In a federated system, nicknames for relational data sources or local tables can be used as the underlying tables to create a materialized query table. Nicknames for non-relational data sources are not supported. When a nickname is one of the underlying tables, the REFRESH DEFERRED option must be used. System-maintained materialized query tables that reference nicknames are not supported in a partitioned database environment.
- ***Transparent DDL:*** In a federated system, a remote base table can be created, altered, or dropped using DB2 SQL. This capability is known as *transparent DDL*. Before a remote base table can be created on a data source, the federated server must be configured to access that data source. This configuration includes creating the wrapper for the data source, supplying the server definition for the server where the remote base table will be located, and creating the user mappings between the federated server and the data source.

  Transparent DDL does impose some limitations on what can be included in the CREATE TABLE statement:
  – Only columns and a primary key can be created on the remote base table.
  – Specific clauses supported by transparent DDL include:
    - *column-definition* and *unique-constraint* in the *element-list* clause
    - NOT NULL and PRIMARY KEY in the *column-options* clause
    - OPTIONS
  – The remote data source must support:
    - The remote column data types to which the DB2 column data types are mapped

  - The primary key option in the CREATE TABLE statement

  Depending on how the data source responds to requests it does not support, an error might be returned or the request might be ignored.

  When a remote base table is created using transparent DDL, a nickname is automatically created for that remote base table.

- A referential constraint may be defined in such a way that either the parent table or the dependent table is a part of a table hierarchy. In such a case, the effect of the referential constraint is as follows:
  1. Effects of INSERT, UPDATE, and DELETE statements:
     – If a referential constraint exists, in which PT is a parent table and DT is a dependent table, the constraint ensures that for each row of DT (or any of its subtables) that has a non-null foreign key, a row exists in PT (or one of its subtables) with a matching parent key. This rule is enforced against any action that affects a row of PT or DT, regardless of how that action is initiated.
  2. Effects of DROP TABLE statements:
     – for referential constraints in which the dropped table is the parent table or dependent table, the constraint is dropped
     – for referential constraints in which a supertable of the dropped table is the parent table the rows of the dropped table are considered to be deleted from the supertable. The referential constraint is checked and its delete rule is invoked for each of the deleted rows.
     – for referential constraints in which a supertable of the dropped table is the dependent table, the constraint is not checked. Deletion of a row from a dependent table cannot result in violation of a referential constraint.
- ***Privileges:*** When any table is created, the definer of the table is granted CONTROL privilege. When a subtable is created, the SELECT privilege that each user or group has on the immediate supertable is automatically granted on the subtable with the table definer as the grantor.
- ***Row Size:*** The maximum number of bytes allowed in the row of a table is dependent on the page size of the table space in which the table is created (*tablspace-name1*). The following list shows the row size limit and number of columns limit associated with each table space page size.

Table 141. Limits for Number of Columns and Row Size in Each Table Space Page Size

Page Size	Row Size Limit	Column Count Limit
4K	4 005	500
8K	8 101	1 012
16K	16 293	1 012
32K	32 677	1 012

  The actual number of columns for a table may be further limited by the following formula:

```
Total Columns * 8 + Number of LOB Columns * 12 <= row size limit for page size.
```

- ***Byte Counts:*** The following table contains the byte counts of columns by data type. This is used to calculate the row size. The byte counts depend on whether or not VALUE COMPRESSION is active. When VALUE COMPRESSION is not active, the byte counts also depend on whether or not the column is nullable.

  If a table is based on a structured type, an additional 4 bytes of overhead is reserved to identify rows of subtables, regardless of whether or not subtables are defined. Additional subtable columns must be considered nullable for byte count purposes, even if defined as not nullable.

Table 142. Byte Counts of Columns by Data Type

Data type	VALUE COMPRESSION is active[1]	VALUE COMPRESSION is not active	
		Column is nullable	**Column is not nullable**
SMALLINT	4	3	2
INTEGER	6	5	4
BIGINT	10	9	8
REAL	6	5	4
DOUBLE	10	9	8
DECIMAL	The integral part of (*p*/2)+3, where *p* is the precision	The integral part of (*p*/2)+2, where *p* is the precision	The integral part of (*p*/2)+1, where *p* is the precision
CHAR(*n*)	*n*+2	*n*+1	*n*
VARCHAR(*n*)	*n*+2	*n*+5 (within a table)	*n*+4 (within a table)
LONG VARCHAR	22	25	24
GRAPHIC(*n*)	*n**2+2	*n**2+1	*n**2
VARGRAPHIC(*n*)	*n**2+2	*n**2+5 (within a table)	*n**2+4 (within a table)
LONG VARGRAPHIC	22	25	24
DATE	6	5	4
TIME	5	4	3
TIMESTAMP	12	11	10
XML	82	85	84
Maximum LOB[2] length 1024	70	73	72
Maximum LOB length 8192	94	97	96
Maximum LOB length 65 536	118	121	120
Maximum LOB length 524 000	142	145	144
Maximum LOB length 4 190 000	166	169	168
Maximum LOB length 134 000 000	198	201	200
Maximum LOB length 536 000 000	222	225	224
Maximum LOB length 1 070 000 000	254	257	256
Maximum LOB length 1 470 000 000	278	281	280
Maximum LOB length 2 147 483 647	314	317	316

Table 142. Byte Counts of Columns by Data Type (continued)

Data type	VALUE COMPRESSION is active[1]	VALUE COMPRESSION is not active	
		Column is nullable	Column is not nullable

[1] There is an additional 2 bytes of storage used by each row when VALUE COMPRESSION is active for that row.

[2] Each LOB value has a *LOB descriptor* in the base record that points to the location of the actual value. The size of the descriptor varies according to the maximum length defined for the column.

For a *distinct type*, the byte count is equivalent to the length of the source type of the distinct type. For a *reference type*, the byte count is equivalent to the length of the built-in data type on which the reference type is based. For a *structured type*, the byte count is equivalent to the INLINE LENGTH + 4. The INLINE LENGTH is the value specified (or implicitly calculated) for the column in the *column-options* clause.

The row sizes for the following sample tables assume that VALUE COMPRESSION is not specifed:

```
DEPARTMENT 63 (0 + 3 + 33 + 7 +  3 + 17)
ORG        57 (0 + 3 + 19 + 2 + 15 + 18)
```

If VALUE COMPRESSION were to be specified, the row sizes would change to:

```
DEPARTMENT 69 (2 + 5 + 31 + 8 +  5 + 18)
ORG        53 (2 + 4 + 16 + 4 + 12 + 15)
```

- ***Storage Byte Counts:*** The following table contains the storage byte counts of columns by data type for data values. The byte counts depend on whether or not VALUE COMPRESSION is active. When VALUE COMPRESSION is not active, the byte counts also depend on whether or not the column is nullable. The values in the table represent the amount of storage (in bytes) that is used to store the value.

Table 143. Storage Byte Counts Based on Row Format, Data Type, and Data Value

Data value →	NULL	NULL	zero-length	system default[2]	all other data values	all other data values	all other data values
VALUE COMPRES-SION →	not active	active[1]	active[1]	active[1]	not active	not active	active[1]
Column nullability →	nullable	nullable	n/a	n/a	nullable	not nullable	n/a
Data type							
SMALLINT	3	3	-	3	3	2	4
INTEGER	5	3	-	3	5	4	6
BIGINT	9	3	-	3	9	8	10
REAL	5	3	-	3	5	4	6
DOUBLE	9	3	-	3	9	8	10
DECIMAL	The integral part of (*p*/2)+2, where *p* is the precision	3	-	3	The integral part of (*p*/2)+2, where *p* is the precision	The integral part of (*p*/2)+1, where *p* is the precision	The integral part of (*p*/2)+3, where *p* is the precision
CHAR(*n*)	*n*+1	3	-	3	*n*+1	*n*	*n*+2
VARCHAR(*n*)	5	3	2	2	*N*+5, where *N* is the number of bytes in the data	*N*+4, where *N* is the number of bytes in the data	*N*+2, where *N* is the number of bytes in the data
LONG VARCHAR	5	3	2	2	25	24	22
GRAPHIC(*n*)	*n**2+1	3	-	3	*n**2+1	*n**2	*n**2+2

Table 143. Storage Byte Counts Based on Row Format, Data Type, and Data Value (continued)

Data value →	NULL	NULL	zero-length	system default[2]	all other data values	all other data values	all other data values
VALUE COMPRES-SION →	not active	active[1]	active[1]	active[1]	not active	not active	active[1]
Column nullability →	nullable	nullable	n/a	n/a	nullable	not nullable	n/a
Data type							
VARGRAPHIC(*n*)	5	3	2	2	*N**2+5, where *N* is the number of bytes in the data	*N**2+4, where *N* is the number of bytes in the data	*N**2+2, where *N* is the number of bytes in the data
LONG VARGRAPHIC	5	3	2	2	25	24	22
DATE	5	3	-	-	5	4	6
TIME	4	3	-	-	4	3	5
TIMESTAMP	11	3	-	-	11	10	12
Maximum LOB[2] length 1024	5	3	2	2	(60 to 68)+5	(60 to 68)+4	(60 to 68)+2
Maximum LOB length 8192	5	3	2	2	(60 to 92)+5	(60 to 92)+4	(60 to 92)+2
Maximum LOB length 65 536	5	3	2	2	(60 to 116)+5	(60 to 116)+4	(60 to 116)+2
Maximum LOB length 524 000	5	3	2	2	(60 to 140)+5	(60 to 140)+4	(60 to 140)+2
Maximum LOB length 4 190 000	5	3	2	2	(60 to 164)+5	(60 to 164)+4	(60 to 164)+2
Maximum LOB length 134 000 000	5	3	2	2	(60 to 196)+5	(60 to 196)+4	(60 to 196)+2
Maximum LOB length 536 000 000	5	3	2	2	(60 to 220)+5	(60 to 220)+4	(60 to 220)+2
Maximum LOB length 1 070 000 000	5	3	2	2	(60 to 252)+5	(60 to 252)+4	(60 to 252)+2
Maximum LOB length 1 470 000 000	5	3	2	2	(60 to 276)+5	(60 to 276)+4	(60 to 276)+2
Maximum LOB length 2 147 483 647	5	3	2	2	(60 to 312)+5	(60 to 312)+4	(60 to 312)+2
XML	5	3	-	-	85	84	82

[1] There is an additional 2 bytes of storage used by each row when VALUE COMPRESSION is active for that row.

[2] When COMPRESS SYSTEM DEFAULT is specified for the column.

- ***Dimension Columns:*** Because each distinct value of a dimension column is assigned to a different block of the table, clustering on an expression may be desirable, such as ″INTEGER(ORDER_DATE)/100″. In this case, a generated column can be defined for the table, and this generated column may then be used in the ORGANIZE BY DIMENSIONS clause. If the expression is monotonic with respect to a column of the table, DB2 may use the dimension index to satisfy range predicates on that column. For example, if the expression is simply *column-name + some-positive-constant*, it is monotonic increasing. User-defined functions, certain built-in functions, and using more than one column in an expression, prevent monotonicity or its detection.

Dimensions involving generated columns whose expressions are non-monotonic, or whose monotonicity cannot be determined, can still be created, but range queries along slice or cell boundaries of these dimensions are not supported. Equality and IN predicates *can* be processed by slices or cells.

A generated column is monotonic if the following is true with respect to the generating function, fn:

– Monotonic increasing.

  For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>fn(x1). For example:

```
SALARY - 10000
```

– Monotonic decreasing.

  For every possible pair of values x1 and x2, if x2>x1, then fn(x2)<fn(x1). For example:

```
-SALARY
```

– Monotonic non-decreasing.

  For every possible pair of values x1 and x2, if x2>x1, then fn(x2)>=fn(x1). For example:

```
SALARY/1000
```

– Monotonic non-increasing.

  For every possible pair of values x1 and x2, if x2>x1, then fn(x2)<=fn(x1). For example:

```
-SALARY/1000
```

The expression ″PRICE*DISCOUNT″ is not monotonic, because it involves more than one column of the table.

- ***Range-clustered tables:*** Organizing a table by key sequence is effective for certain types of tables. The table should have an integer key that is tightly clustered (dense) over the range of possible values. The columns of this integer key must not be nullable, and the key should logically be the primary key of the table. The organization of a range-clustered table precludes the need for a separate unique index object, providing direct access to the row for a specified key value, or a range of rows for a specified range of key values. The allocation of all the space for the complete set of rows in the defined key sequence range is done during table creation, and must be considered when defining a range-clustered table. The storage space is not available for any other use, even though the rows are initially marked deleted. If the full key sequence range will be populated with data only over a long period of time, this table organization may not be an appropriate choice.
- A table can have at most one security policy.
- DB2 enforces referential integrity constraints that are defined on protected tables. Constraints violations in this case can be difficult to debug, because DB2 will not allow you to see what row has caused a violation if you do not have the appropriate security label or exemptions credentials.
- ***Security and replication:*** Replication can cause data rows from a protected table to be replicated outside of the database. Care must be taken when setting up replication for a protected table, because DB2 cannot protect data that is outside of the database.

- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - The CONSTRAINT keyword can be omitted from a *column-definition* defining a references-clause
    - *constraint-name* can be specified following FOREIGN KEY (without the CONSTRAINT keyword)
    - SUMMARY can optionally be specified after CREATE
    - DEFINITION ONLY can be specified in place of WITH NO DATA
    - The PARTITIONING KEY clause can be specified in place of the DISTRIBUTE BY clause
  - For compatibility with previous versions of DB2, and for consistency:
    - A comma can be used to separate multiple options in the *identity-options* clause
  - For compatibility with DB2 UDB for OS/390 and z/OS:
    - The following syntax is accepted as the default behavior:
      - IN database-name.tablespace-name
      - IN DATABASE database-name
      - FOR MIXED DATA
      - FOR SBCS DATA
    - PART can be specified in place of PARTITION
    - PARTITION *partition-number* can be specified instead of PARTITION *partition-name*. A *partition-number* must not identify a partition that was previously specified in the CREATE TABLE statement. If a *partition-number* is not specified, a unique partition number is generated by the database manager.
    - VALUES can be specified in place of ENDING AT
  - The following syntax is also supported:
    - NOMINVALUE, NOMAXVALUE, NOCYCLE, NOCACHE, and NOORDER

Examples:

Example 1: Create table TDEPT in the DEPARTX table space. DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT are column names. CHAR means the column will contain character data. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. The primary key consists of the column DEPTNO.

```
CREATE TABLE TDEPT
  (DEPTNO   CHAR(3)     NOT NULL,
   DEPTNAME VARCHAR(36) NOT NULL,
   MGRNO    CHAR(6),
   ADMRDEPT CHAR(3)     NOT NULL,
   PRIMARY KEY(DEPTNO))
IN DEPARTX
```

Example 2: Create table PROJ in the SCHED table space. PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTAFF, PRSTDATE, PRENDATE, and MAJPROJ are column names. CHAR means the column will contain character data. DECIMAL means the column will contain packed decimal data. 5,2 means the following: 5 indicates the number of decimal digits, and 2 indicates the number of digits to the right of the decimal point. NOT NULL means that the column cannot contain a null value. VARCHAR means the column will contain varying-length character data. DATE means the column will contain date information in a three-part format (year, month, and day).

```
CREATE TABLE PROJ
  (PROJNO   CHAR(6)      NOT NULL,
   PROJNAME VARCHAR(24)  NOT NULL,
   DEPTNO   CHAR(3)      NOT NULL,
   RESPEMP  CHAR(6)      NOT NULL,
   PRSTAFF  DECIMAL(5,2)         ,
   PRSTDATE DATE                 ,
   PRENDATE DATE                 ,
   MAJPROJ  CHAR(6)      NOT NULL)
IN SCHED
```

Example 3: Create a table called EMPLOYEE_SALARY where any unknown salary is considered 0. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the *IN tablespace-name1* clause.

```
CREATE TABLE EMPLOYEE_SALARY
  (DEPTNO   CHAR(3)      NOT NULL,
   DEPTNAME VARCHAR(36)  NOT NULL,
   EMPNO    CHAR(6)      NOT NULL,
   SALARY   DECIMAL(9,2) NOT NULL WITH DEFAULT)
```

Example 4: Create distinct types for total salary and miles and use them for columns of a table created in the default table space. In a dynamic SQL statement assume the CURRENT SCHEMA special register is JOHNDOE and the CURRENT PATH is the default ("SYSIBM","SYSFUN","JOHNDOE").

If a value for SALARY is not specified it must be set to 0 and if a value for LIVING_DIST is not specified it must to set to 1 mile.

```
CREATE DISTINCT TYPE JOHNDOE.T_SALARY AS INTEGER WITH COMPARISONS

CREATE DISTINCT TYPE JOHNDOE.MILES  AS FLOAT WITH COMPARISONS

CREATE TABLE EMPLOYEE
  (ID          INTEGER NOT NULL,
   NAME        CHAR (30),
   SALARY      T_SALARY NOT NULL WITH DEFAULT,
   LIVING_DIST MILES    DEFAULT MILES(1) )
```

Example 5: Create distinct types for image and audio and use them for columns of a table. No table space is specified, so that the table will be created in a table space selected by the system based on the rules described for the IN *tablespace-name1* clause. Assume the CURRENT PATH is the default.

```
CREATE DISTINCT TYPE IMAGE AS BLOB (10M)

CREATE DISTINCT TYPE AUDIO AS BLOB (1G)

CREATE TABLE PERSON
  (SSN    INTEGER NOT NULL,
   NAME   CHAR (30),
   VOICE  AUDIO,
   PHOTO  IMAGE)
```

Example 6: Create table EMPLOYEE in the HUMRES table space. The constraints defined on the table are the following:

- The values of department number must lie in the range 10 to 100.
- The job of an employee can only be either 'Sales', 'Mgr' or 'Clerk'.

- Every employee that has been with the company since 1986 must make more than $40,500.

Note: If the columns included in the check constraints are nullable they could also be NULL.

```
CREATE TABLE EMPLOYEE
  (ID         SMALLINT NOT NULL,
   NAME       VARCHAR(9),
   DEPT       SMALLINT CHECK (DEPT BETWEEN 10 AND 100),
   JOB        CHAR(5) CHECK (JOB IN ('Sales','Mgr','Clerk')),
   HIREDATE   DATE,
   SALARY     DECIMAL(7,2),
   COMM       DECIMAL(7,2),
   PRIMARY KEY (ID),
   CONSTRAINT  YEARSAL CHECK (YEAR(HIREDATE) > 1986
     OR SALARY > 40500)
  )
  IN HUMRES
```

Example 7: Create a table that is wholly contained in the PAYROLL table space.

```
CREATE TABLE EMPLOYEE .....
  IN PAYROLL
```

Example 8: Create a table with its data part in ACCOUNTING and its index part in ACCOUNT_IDX.

```
CREATE TABLE SALARY.....
  IN ACCOUNTING INDEX IN ACCOUNT_IDX
```

Example 9: Create a table and log SQL changes in the default format.

```
CREATE TABLE SALARY1 .....
```

or

```
CREATE TABLE SALARY1 .....
  DATA CAPTURE NONE
```

Example 10: Create a table and log SQL changes in an expanded format.

```
CREATE TABLE SALARY2 .....
  DATA CAPTURE CHANGES
```

Example 11: Create a table EMP_ACT in the SCHED table space. EMPNO, PROJNO, ACTNO, EMPTIME, EMSTDATE, and EMENDATE are column names. Constraints defined on the table are:

- The value for the set of columns, EMPNO, PROJNO, and ACTNO, in any row must be unique.
- The value of PROJNO must match an existing value for the PROJNO column in the PROJECT table and if the project is deleted all rows referring to the project in EMP_ACT should also be deleted.

```
CREATE TABLE EMP_ACT
  (EMPNO      CHAR(6) NOT NULL,
   PROJNO     CHAR(6) NOT NULL,
   ACTNO      SMALLINT NOT NULL,
   EMPTIME    DECIMAL(5,2),
   EMSTDATE   DATE,
   EMENDATE   DATE,
   CONSTRAINT EMP_ACT_UNIQ UNIQUE (EMPNO,PROJNO,ACTNO),
   CONSTRAINT FK_ACT_PROJ FOREIGN KEY (PROJNO)
                      REFERENCES PROJECT (PROJNO) ON DELETE CASCADE
  )
  IN SCHED
```

A unique index called EMP_ACT_UNIQ is automatically created in the same schema to enforce the unique constraint.

Example 12: Create a table that is to hold information about famous goals for the ice hockey hall of fame. The table will list information about the player who scored the goal, the goaltender against who it was scored, the date and place, and a description. The description column is nullable.

```
CREATE TABLE HOCKEY_GOALS
  ( BY_PLAYER      VARCHAR(30)   NOT NULL,
    BY_TEAM        VARCHAR(30)   NOT NULL,
    AGAINST_PLAYER VARCHAR(30)   NOT NULL,
    AGAINST_TEAM   VARCHAR(30)   NOT NULL,
    DATE_OF_GOAL   DATE          NOT NULL,
    DESCRIPTION    CLOB(5000) )
```

Example 13: Suppose an exception table is needed for the EMPLOYEE table. One can be created using the following statement.

```
CREATE TABLE EXCEPTION_EMPLOYEE AS
  (SELECT EMPLOYEE.*,
    CURRENT TIMESTAMP AS TIMESTAMP,
    CAST ('' AS CLOB(32K)) AS MSG
  FROM EMPLOYEE
  ) WITH NO DATA
```

Example 14: Given the following table spaces with the indicated attributes:

```
TBSPACE            PAGESIZE    USER   USERAUTH
------------------ ----------- ------ --------
DEPT4K                    4096 BOBBY  Y
PUBLIC4K                  4096 PUBLIC Y
DEPT8K                    8192 BOBBY  Y
DEPT8K                    8192 RICK   Y
PUBLIC8K                  8192 PUBLIC Y
```

- If RICK creates the following table, it is placed in table space PUBLIC4K since the byte count is less than 4005; but if BOBBY creates the same table, it is placed in table space DEPT4K, since BOBBY has USE privilege because of an explicit grant:

  ```
  CREATE TABLE DOCUMENTS
    (SUMMARY    VARCHAR(1000),
     REPORT     VARCHAR(2000))
  ```

- If BOBBY creates the following table, it is placed in table space DEPT8K since the byte count is greater than 4005, and BOBBY has USE privilege because of an explicit grant. However, if DUNCAN creates the same table, it is placed in table space PUBLIC8K, since DUNCAN has no specific privileges:

  ```
  CREATE TABLE CURRICULUM
    (SUMMARY    VARCHAR(1000),
     REPORT     VARCHAR(2000),
     EXERCISES  VARCHAR(1500))
  ```

Example 15: Create a table with a LEAD column defined with the structured type EMP. Specify an INLINE LENGTH of 300 bytes for the LEAD column, indicating that any instances of LEAD that cannot fit within the 300 bytes are stored outside the table (separately from the base table row, similar to the way LOB values are handled).

```
CREATE TABLE PROJECTS (PID INTEGER,
  LEAD EMP INLINE LENGTH 300,
  STARTDATE DATE,
          ...)
```

Example 16: Create a table DEPT with five columns named DEPTNO, DEPTNAME, MGRNO, ADMRDEPT, and LOCATION. Column DEPT is to be defined as an IDENTITY column such that DB2 will always generate a value for it. The values for the DEPT column should begin with 500 and increment by 1.

```
CREATE TABLE DEPT
  (DEPTNO     SMALLINT      NOT NULL
                 GENERATED ALWAYS AS IDENTITY
                   (START WITH 500, INCREMENT BY 1),
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30))
```

Example 17: Create a SALES table that is distributed on the YEAR column, and that has dimensions on the REGION and YEAR columns. Data will be distributed across database partitions according to hashed values of the YEAR column. On each database partition, data will be organized into extents based on unique combinations of values of the REGION and YEAR columns on those database partitions.

```
CREATE TABLE SALES
  (CUSTOMER   VARCHAR(80),
   REGION     CHAR(5),
   YEAR       INTEGER)
DISTRIBUTE BY HASH (YEAR)
ORGANIZE BY DIMENSIONS (REGION, YEAR)
```

Example 18: Create a SALES table with a PURCHASEYEARMONTH column that is generated from the PURCHASEDATE column. Use an expression to create a column that is monotonic with respect to the original PURCHASEDATE column, and is therefore suitable for use as a dimension. The table is distributed on the REGION column, and organized within each database partition into extents according to the PURCHASEYEARMONTH column; that is, different regions will be on different database partitions, and different purchase months will belong to different cells (or sets of extents) within those database partitions.

```
CREATE TABLE SALES
  (CUSTOMER            VARCHAR(80),
   REGION              CHAR(5),
   PURCHASEDATE        DATE,
   PURCHASEYEARMONTH   INTEGER
                          GENERATED ALWAYS AS (INTEGER(PURCHASEDATE)/100))
DISTRIBUTE BY HASH (REGION)
ORGANIZE BY DIMENSIONS (PURCHASEYEARMONTH)
```

Example 19: Create a CUSTOMER table with a CUSTOMERNUMDIM column that is generated from the CUSTOMERNUM column. Use an expression to create a column that is monotonic with respect to the original CUSTOMERNUM column, and is therefore suitable for use as a dimension. The table is organized into cells according to the CUSTOMERNUMDIM column, so that there is a different cell in the table for every 50 customers. If a unique index were created on CUSTOMERNUM, customer numbers would be clustered in such a way that each set of 50 values would be found in a particular set of extents in the table.

```
CREATE TABLE CUSTOMER
  (CUSTOMERNUM      INTEGER,
   CUSTOMERNAME     VARCHAR(80),
   ADDRESS          VARCHAR(200),
   CITY             VARCHAR(50),
   COUNTRY          VARCHAR(50),
   CODE             VARCHAR(15),
   CUSTOMERNUMDIM   INTEGER
                       GENERATED ALWAYS AS (CUSTOMERNUM/50))
ORGANIZE BY DIMENSIONS (CUSTOMERNUMDIM)
```

Example 20: Create a remote base table called EMPLOYEE on the Oracle server, ORASERVER. A nickname, named EMPLOYEE, which refers to this newly created remote base table, will also automatically be created.

```
  CREATE TABLE EMPLOYEE
    (EMP_NO      CHAR(6)       NOT NULL,
     FIRST_NAME  VARCHAR(12)   NOT NULL,
     MID_INT     CHAR(1)       NOT NULL,
     LAST_NAME   VARCHAR(15)   NOT NULL,
     HIRE_DATE   DATE,
     JOB         CHAR(8),
     SALARY      DECIMAL(9,2),
     PRIMARY KEY (EMP_NO))
  OPTIONS
    (REMOTE_SERVER 'ORASERVER',
     REMOTE_SCHEMA 'J15USER1',
     REMOTE_TABNAME 'EMPLOYEE')
```

The following CREATE TABLE statements show how to specify the table name, or the table name and the explicit remote base table name, to get the desired case. The lowercase identifier, employee, is used to illustrate the implicit folding of identifiers.

Create a remote base table called EMPLOYEE (uppercase characters) on an Informix® server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```
  CREATE TABLE employee
    (EMP_NO CHAR(6) NOT NULL,
     ...)
  OPTIONS
    (REMOTE_SERVER 'INFX_SERVER')
```

If the REMOTE_TABNAME option is not specified, and *table-name* is not delimited, the remote base table name will be in uppercase characters, even if the remote data source normally stores names in lowercase characters.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named EMPLOYEE (uppercase characters) on that table:

```
  CREATE TABLE employee
    (EMP_NO CHAR(6) NOT NULL,
     ...)
  OPTIONS
    (REMOTE_SERVER 'INFX_SERVER',
     REMOTE_TABNAME 'employee')
```

When creating a table at a remote data source that supports delimited identifiers, use the REMOTE_TABNAME option and a character string constant that specifies the table name in the desired case.

Create a remote base table called employee (lowercase characters) on an Informix server, and create a nickname named employee (lowercase characters) on that table:

```
CREATE TABLE "employee"
  (EMP_NO CHAR(6) NOT NULL,
   ...)
OPTIONS
  (REMOTE_SERVER 'INFX_SERVER')
```

If the REMOTE_TABNAME option is not specified, and *table-name* is delimited, the remote base table name will be identical to *table-name*.

Example 21: Create a range-clustered table that can be used to locate a student using a student ID. For each student record, include the school ID, program ID, student number, student ID, student first name, student last name, and student grade point average (GPA).

```
CREATE TABLE STUDENTS
  (SCHOOL_ID     INTEGER   NOT NULL,
   PROGRAM_ID    INTEGER   NOT NULL,
   STUDENT_NUM   INTEGER   NOT NULL,
   STUDENT_ID    INTEGER   NOT NULL,
   FIRST_NAME    CHAR(30),
   LAST_NAME     CHAR(30),
   GPA           DOUBLE)
ORGANIZE BY KEY SEQUENCE
  (STUDENT_ID
    STARTING FROM 1
    ENDING AT 1000000)
  DISALLOW OVERFLOW
```

The size of each record is the sum of the columns, plus alignment, plus the range-clustered table row header. In this case, the row size is 98 bytes: 4 + 4 + 4 + 4 + 30 + 30 + 8 + 3 (for nullable columns) + 1 (for alignment) + 10 (for the header). With a 4-KB page size (or 4096 bytes), after accounting for page overhead, there are 4038 bytes available, enough room for 41 records per page. Allowing for 1 million student records, there is a need for (1 million divided by 41 records per page) 24 391 pages. With two additional pages for table overhead, the final number of 4-KB pages that are allocated when the table is created is 24 393.

Example 22: Create a table named DEPARTMENT with a functional dependency that has no specified constraint name.

```
CREATE TABLE DEPARTMENT
  (DEPTNO     SMALLINT      NOT NULL,
   DEPTNAME   VARCHAR(36)   NOT NULL,
   MGRNO      CHAR(6),
   ADMRDEPT   SMALLINT      NOT NULL,
   LOCATION   CHAR(30),
  CHECK (DEPTNAME DETERMINED BY DEPTNO) NOT ENFORCED)
```

Example 23: Create a protected table with row-level granularity.

```
CREATE TABLE TOASTMASTERS
  (PERFORMANCE DB2SECURITYLABEL,
   POINTS      INTEGER,
   NAME        VARCHAR(50))
  SECURITY POLICY CONTRIBUTIONS
```

Example 24: Create a protected table with column-level granularity.

```
CREATE TABLE TOASTMASTERS
  (PERFORMANCE CHAR(8),
   POINTS      INTEGER COLUMN SECURED WITH CLUBPOSITION,
   NAME        VARCHAR(50))
  SECURITY POLICY CONTRIBUTIONS
```

Example 25: Create a protected table with both row-level and column-level granularities.

```
CREATE TABLE TOASTMASTERS
  (PERFORMANCE DB2SECURITYLABEL,
   POINTS      INTEGER COLUMN SECURED WITH CLUBPOSITION,
   NAME        VARCHAR(50))
  SECURITY POLICY CONTRIBUTIONS
```

Example 26: Large objects for a partitioned table reside, by default, in the same table space as the data. This default behavior can be overridden by using the LONG IN clause to specify one or more table spaces for the large objects. Create a table named DOCUMENTS whose large object data is to be stored (in a round-robin fashion for each data partition) in table spaces TBSP1 and TBSP2.

```
CREATE TABLE DOCUMENTS
  (ID INTEGER,
   CONTENTS CLOB)
  LONG IN TBSP1, TBSP2
  PARTITION BY RANGE (ID)
    (STARTING 1 ENDING 1000 EVERY 100)
```

Alternatively, use the long form of the syntax to explicitly identify a large table space for each data partition. In this example, the CLOB data for the first data partition is placed in TBSP3, and the CLOB data for the remaining data partitions is spread across TBSP1 and TBSP2 in a round-robin fashion.

```
CREATE TABLE DOCUMENTS
  (ID INTEGER,
   CONTENTS CLOB)
  LONG IN TBSP1, TBSP2
  PARTITION BY RANGE (ID)
    (STARTING 1 ENDING 100 LONG IN TBSP3,
     STARTING 101 ENDING 1000 EVERY 100)
```

Example 27: Create a partitioned table named ACCESSNUMBERS having two data partitions. The row (10, NULL) is to be placed in the first partition, and the row (NULL, 100) is to be placed in the second (last) data partition.

```
CREATE TABLE ACCESSNUMBERS
  (AREA INTEGER,
   EXCHANGE INTEGER)
  PARTITION BY RANGE (AREA NULLS LAST, EXCHANGE NULLS FIRST)
    (STARTING (1,1) ENDING (10,100),
     STARTING (11,1) ENDING (MAXVALUE,MAXVALUE))
```

Because null values in the second column are sorted first, the row (11, NULL) would sort below the low boundary of the last data partition (11, 1); attempting to insert this row returns an error. The row (12, NULL) would fall within the last data partition.

Example 28: Create a table named RATIO having a single data partition and partitioning column PERCENT.

```
CREATE TABLE RATIO
  (PERCENT INTEGER)
  PARTITION BY RANGE (PERCENT)
    (STARTING (MINVALUE) ENDING (MAXVALUE))
```

This table definition allows any integer value for column PERCENT to be inserted. The following definition for the RATIO table allows any integer value between 1 and 100 inclusive to be inserted into column PERCENT.

```
CREATE TABLE RATIO
  (PERCENT INTEGER)
  PARTITION BY RANGE (PERCENT)
    (STARTING 0 EXCLUSIVE ENDING 100 INCLUSIVE)
```

Example 29: Create a table named MYDOCS with two columns: one is an identifier, and the other stores XML documents.

```
CREATE TABLE MYDOCS
  (ID INTEGER,
   DOC XML)
IN HLTBSPACE
```

Example 30: Create a table named NOTES with four columns, including one for storing XML-based notes.

```
CREATE TABLE NOTES
  (ID          INTEGER,
   DESCRIPTION VARCHAR(255),
   CREATED     TIMESTAMP,
   NOTE        XML)
```

Related concepts:

- "Data organization schemes in DB2 and Informix databases" in *Administration Guide: Planning*
- "Multidimensional clustering tables" in *Administration Guide: Planning*
- "Partitioned tables" in *Administration Guide: Planning*
- "Table creation" in *Administration Guide: Implementation*
- "Table partitioning" in *Administration Guide: Planning*
- "Table partitioning keys" in *Administration Guide: Planning*

Related tasks:

- "Creating and populating a table" on page 306
- "Recovering a dropped table" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "Assignments and comparisons" in *SQL Reference, Volume 1*
- "Database partition-compatible data types" in *SQL Reference, Volume 1*
- "SQL and XQuery limits" in *SQL Reference, Volume 1*
- "ALTER TABLE " on page 854
- "CREATE TABLESPACE " on page 1021
- "DECLARE GLOBAL TEMPORARY TABLE statement" in *SQL Reference, Volume 2*
- "Subselect" on page 1211

Related samples:

- "dtudt.c -- How to create, use, and drop user-defined distinct types."
- "tbconstr.c -- How to work with constraints associated with tables"
- "tbcreate.c -- How to create, alter and drop tables"
- "dtudt.sqc -- How to create, use, and drop user-defined distinct types (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbcreate.sqc -- How to create and drop tables (C)"
- "tbident.sqc -- How to use identity columns (C)"

- “tbtrig.sqc -- How to use a trigger on a table (C)”
- “dtudt.sqC -- How to create, use, and drop user-defined distinct types (C++)”
- “tbconstr.sqC -- How to create, use, and drop constraints (C++)”
- “tbcreate.sqC -- How to create and drop tables (C++)”
- “tbtrig.sqC -- How to use a trigger on a table (C++)”
- “DtUdt.java -- How to create, use and drop user defined distinct types (JDBC)”
- “TbConstr.java -- How to create, use and drop constraints (JDBC)”
- “TbCreate.java -- How to create and drop tables (JDBC)”
- “TbGenCol.java -- How to use generated columns (JDBC)”
- “TbIdent.java -- How to use Identity Columns (JDBC)”
- “TbTrig.java -- How to use triggers (JDBC)”
- “DtUdt.sqlj -- How to create, use and drop user defined distinct types (SQLj)”
- “TbConstr.sqlj -- How to create, use and drop constraints (SQLj)”
- “TbCreate.sqlj -- How to create and drop tables (SQLj)”
- “TbIdent.sqlj -- How to use Identity Columns (SQLj)”
- “TbTrig.sqlj -- How to use triggers (SQLj)”
- “impexp.sqb -- Export and import tables with table data (MF COBOL)”

CREATE TABLESPACE

The CREATE TABLESPACE statement defines a new table space within the database, assigns containers to the table space, and records the table space definition and attributes in the catalog.

Invocation:

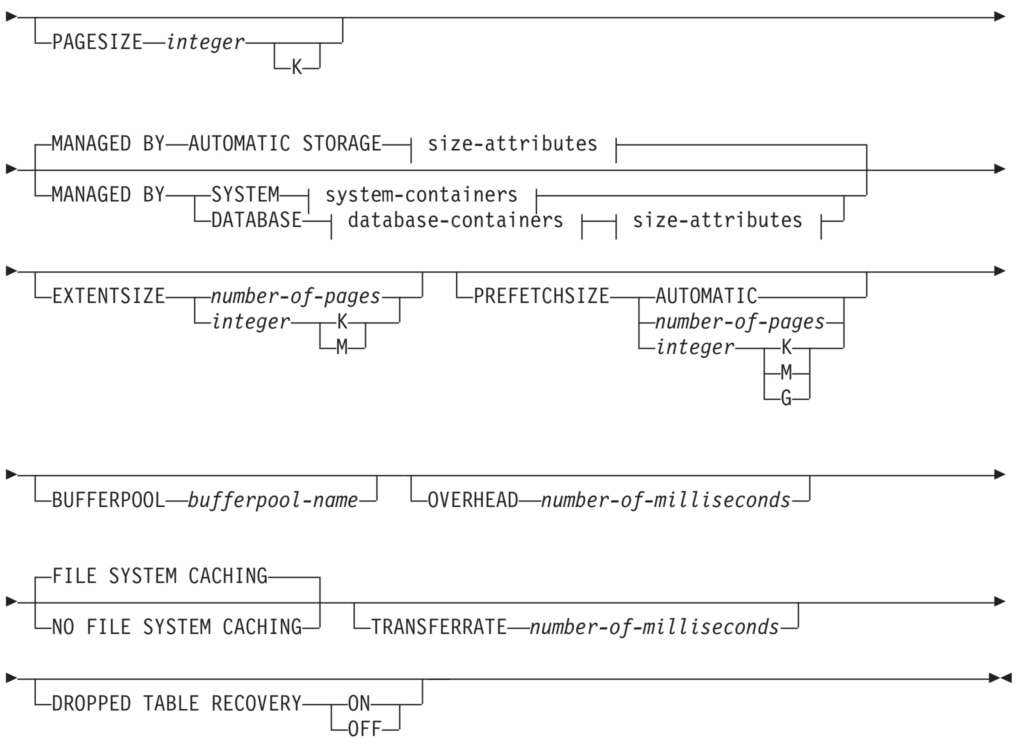
This statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

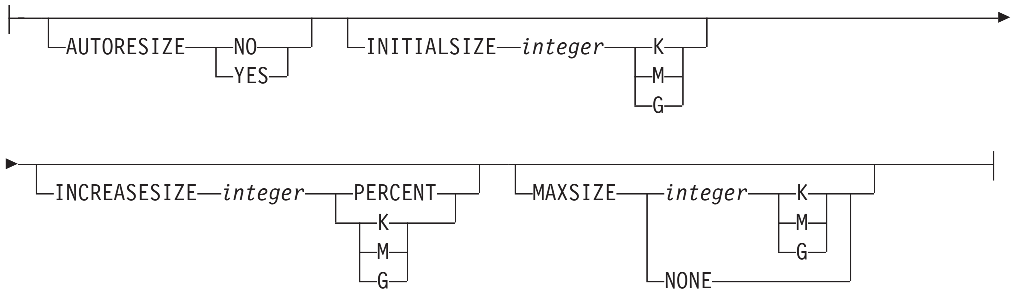
The privileges held by the authorization ID of the statement must include SYSCTRL or SYSADM authority.

Syntax:

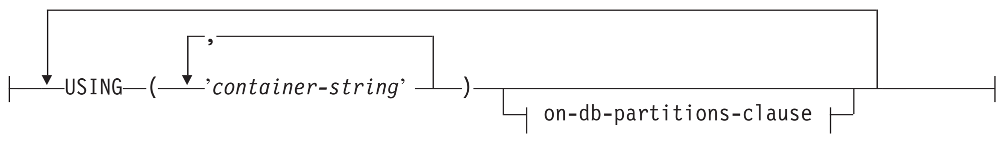
```
►►──CREATE──┬──────────────────────────┬──TABLESPACE──tablespace-name──────────────►
            ├─LARGE────────────────────┤
            ├─REGULAR──────────────────┤
            │ ┌─SYSTEM─┐               │
            └─┴─USER───┴──TEMPORARY────┘

►──┬────────────────────────────────────────────────────────────┬──────────────────►
   │      ┌─DATABASE PARTITION GROUP─┐                          │
   └─IN───┴──────────────────────────┴──db-partition-group-name─┘
```



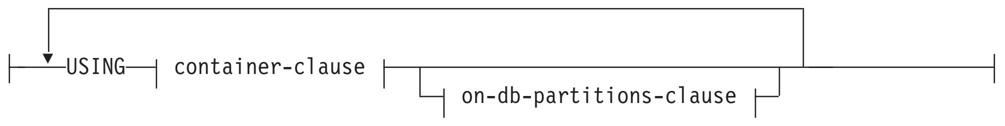
size-attributes:



system-containers:



database-containers:



container-clause:

```
        ┌─,──────────────────────────────────────────────────────┐
        ▼                                                        │
├──(──┬─FILE───┬──'container-string'──┬─number-of-pages─────────┬┴──)──────────────┤
      └─DEVICE─┘                      └─integer──┬─K─┬──────────┘
                                                 ├─M─┤
                                                 └─G─┘
```

on-db-partitions-clause:

```
├──ON──┬─DBPARTITIONNUM──┬──────────────────────────────────────────────────────►
       └─DBPARTITIONNUMS─┘

      ┌─,─────────────────────────────────────────────────────┐
      ▼                                                       │
►─(──db-partition-number1──┬─────────────────────────────┬────┴──)─────────────┤
                           └─TO──db-partition-number2─────┘
```

Description:

LARGE, REGULAR, SYSTEM TEMPORARY, or **USER TEMPORARY**
Specifies the type of table space that is to be created. If no type is specified, the default is determined by the MANAGED BY clause.

LARGE
Stores all permanent data. This type is only allowed on database managed space (DMS) table spaces. It is also the default type for DMS table spaces when no type is specified. When a table is placed in a large table space:

- The table can be larger than a table in a regular table space. For details on table and table space limits, see "SQL limits".
- The table can support more than 255 rows per data page, which can improve space utilization on data pages.
- Indexes that are defined on the table will require an additional 2 bytes per row entry, compared to indexes defined on a table that resides in a regular table space.

REGULAR
Stores all permanent data. This type applies to both DMS and SMS table spaces. This is the only type allowed for SMS table spaces, and it is also the default type for SMS table spaces when no type is specified.

SYSTEM TEMPORARY
Stores temporary tables, work areas used by the database manager to perform operations such as sorts or joins. A database must always have at least one SYSTEM TEMPORARY table space, because temporary tables can only be stored in such a table space. A temporary table space is created automatically when a database is created.

USER TEMPORARY
Stores declared global temporary tables. No user temporary table spaces exist when a database is created. To allow the definition of declared temporary tables, at least one user temporary table space should be created with appropriate USE privileges.

tablespace-name
Names the table space. This is a one-part name. It is an SQL identifier (either ordinary or delimited). The *tablespace-name* must not identify a table space that already exists in the catalog (SQLSTATE 42710). The *tablespace-name* must not begin with the characters 'SYS' (SQLSTATE 42939).

IN DATABASE PARTITION GROUP *db-partition-group-name*
Specifies the database partition group for the table space. The database partition group must exist. The only database partition group that can be specified when creating a SYSTEM TEMPORARY table space is IBMTEMPGROUP. The DATABASE PARTITION GROUP keywords are optional.

If the database partition group is not specified, the default database partition group (IBMDEFAULTGROUP) is used for REGULAR, LARGE, and USER TEMPORARY table spaces. For SYSTEM TEMPORARY table spaces, the default database partition group IBMTEMPGROUP is used.

PAGESIZE *integer* **[K]**
Defines the size of pages used for the table space. The valid values for *integer* without the suffix K are 4 096, 8 192, 16 384, or 32 768. The valid values for *integer* with the suffix K are 4, 8, 16, or 32. Any number of spaces is allowed between *integer* and K, including no space. An error occurs if the page size is not one of these values (SQLSTATE 428DE), or if the page size is not the same as the page size of the buffer pool that is associated with the table space (SQLSTATE 428CB).

The default value is provided by the *pagesize* database configuration parameter, which is set when the database is created.

MANAGED BY AUTOMATIC STORAGE
Specifies that the table space is to be an automatic storage table space. If automatic storage is not defined for the database, an error is returned (SQLSTATE 55060).

An automatic storage table space is created as either a system managed space (SMS) table space or a database managed space (DMS) table space. When DMS is chosen and the type of table space is not specified, the default behavior is to create a large table space. With an automatic storage table space, the database manager determines which containers are to be assigned to the table space, based upon the storage paths that are associated with the database.

size-attributes
Specify the size attributes for an automatic storage table space or a DMS table space that is not an automatic storage table space. SMS table spaces are not auto-resizable.

AUTORESIZE
Specifies whether or not the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled. Auto-resizable table spaces automatically increase in size when they become full. The default is NO for DMS table spaces and YES for automatic storage table spaces.

NO
Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be disabled.

YES
Specifies that the auto-resize capability of a DMS table space or an automatic storage table space is to be enabled.

INITIALSIZE *integer* **K | M | G**
Specifies the initial size, per database partition, of an automatic storage table space. This option is only valid for automatic storage table spaces. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain a consistent size across containers in the table space. Moreover, if the table space is auto-resizable and the initial size is not large enough to contain meta-data that must be added to the new table space, DB2 will continue to extend the table space by INCREASESIZE until there is enough space. If the table space is auto-resizable, but the INITIALSIZE clause is not specified, the database manager determines an appropriate value.

INCREASESIZE *integer* **PERCENT** or **INCREASESIZE** *integer* **K | M | G**
Specifies the amount, per database partition, by which a table space that is enabled for auto-resize will automatically be increased when the table space is full, and a request for space has been made. The integer value must be followed by:

- PERCENT to specify the amount as a percentage of the table space size at the time that a request for space is made. When PERCENT is specified, the integer value must be between 0 and 100 (SQLSTATE 42615).
- K (for kilobytes), M (for megabytes), or G (for gigabytes) to specify the amount in bytes

Note that the actual value used might be slightly smaller or larger than what was specified, because the database manager strives to maintain consistent growth across containers in the table space. If the table space is auto-resizable, but the INCREASESIZE clause is not specified, the database manager determines an appropriate value.

MAXSIZE *integer* **K | M | G** or **MAXSIZE NONE**
Specifies the maximum size to which a table space that is enabled for auto-resize can automatically be increased. If the table space is auto-resizable, but the MAXSIZE clause is not specified, the default is NONE.

integer
Specifies a hard limit on the size, per database partition, to which a DMS table space or an automatic storage table space can automatically be increased. The integer value must be followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). Note that the actual value used might be slightly smaller than what was specified, because the database manager strives to maintain consistent growth across containers in the table space.

NONE
Specifies that the table space is to be allowed to grow to file system capacity, or to the maximum table space size (described in "SQL limits").

MANAGED BY SYSTEM
Specifies that the table space is to be an SMS table space. When the type of table space is not specified, the default behavior is to create a regular table space.

system-containers
Specify the containers for an SMS table space.

USING *('container-string',...)*

For an SMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The *container-string* cannot exceed 240 bytes in length.

Each *container-string* can be an absolute or relative directory name.

The directory name, if not absolute, is relative to the database directory, and can be a path name alias (a symbolic link on UNIX systems) to storage that is not physically associated with the database directory. For example, `<dbdir>/work/c1` could be a symbolic link to a separate file system.

If any component of the directory name does not exist, it is created by the database manager. When a table space is dropped, all components created by the database manager are deleted. If the directory identified by *container-string* exists, it must not contain any files or subdirectories (SQLSTATE 428B2).

The format of *container-string* is dependent on the operating system. On Windows operating systems, an absolute directory path name begins with a drive letter and a colon (:); on UNIX systems, an absolute path name begins with a forward slash (/). A relative path name on any platform does not begin with an operating system-dependent character.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended. An NFS-mounted file system on AIX must be mounted in uninterruptible mode using the -o nointr option.

on-db-partitions-clause

Specifies the database partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the database partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clause*s. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new database partitions added to the database.

MANAGED BY DATABASE

Specifies that the table space is to be a DMS table space. When the type of table space is not specified, the default behavior is to create a large table space.

database-containers

Specify the containers for a DMS table space.

USING

Introduces a container-clause.

container-clause

Specifies the containers for a DMS table space.

(FILE|DEVICE *'container-string' number-of-pages,...***)**

For a DMS table space, identifies one or more containers that will belong to the table space and in which the table space data will be stored. The type of the container (either FILE or DEVICE) and its size (in PAGESIZE pages) are specified. The size can also be specified as an integer value followed by K (for kilobytes), M (for megabytes) or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the pagesize is used to determine the number of pages for the container. A mixture of FILE and DEVICE containers can be specified. The *container-string* cannot exceed 254 bytes in length.

For a FILE container, *container-string* must be an absolute or relative file name. The file name, if not absolute, is relative to the database directory. If any component of the directory name does not exist, it is created by the database manager. If the file does not exist, it will be created and initialized to the specified size by the database manager. When a table space is dropped, all components created by the database manager are deleted.

Note: If the file exists, it is overwritten, and if it is smaller than specified, it is extended. The file will not be truncated if it is larger than specified.

For a DEVICE container, *container-string* must be a device name. The device must already exist.

All containers must be unique across all databases. A container can belong to only one table space. The size of the containers can differ; however, optimal performance is achieved when all containers are the same size. The exact format of *container-string* is dependent on the operating system.

Remote resources (such as LAN-redirected drives or NFS-mounted file systems) are currently only supported when using Network Appliance Filers, IBM iSCSI, IBM Network Attached Storage, Network Appliance iSCSI, NEC iStorage S2100, S2200, or S4100, or NEC Storage NS Series with a Windows DB2 server. Note that NEC Storage NS Series is only supported with the use of an uninterrupted power supply (UPS); continuous UPS (rather than standby) is recommended..

on-db-partitions-clause
Specifies the database partition or partitions on which the containers are created in a partitioned database. If this clause is not specified, then the containers are created on the database partitions in the database partition group that are not explicitly specified in any other *on-db-partitions-clause*. For a SYSTEM TEMPORARY table space defined on database partition group IBMTEMPGROUP, when the *on-db-partitions-clause* is not specified, the containers will also be created on all new database partitions added to the database.

on-db-partitions-clause
Specifies the database partitions on which containers are created in a partitioned database.

ON DBPARTITIONNUMS
Keywords indicating that individual database partitions are specified. DBPARTITIONNUM is a synonym for DBPARTITIONNUMS.

db-partition-number1
Specify a database partition number.

TO *db-partition-number2*
Specify a range of database partition numbers. The value of *db-partition-number2* must be greater than or equal to the value of *db-partition-number1* (SQLSTATE 428A9). Containers are to be created

on each database partition between and including the specified values. A specified database partition must be in the database partition group for the table space.

The database partition specified by number, and every database partition within the specified range of database partitions must exist in the database partition group for the table space (SQLSTATE 42729). A database partition number can only appear explicitly or within a range in exactly one *on-db-partitions-clause* for the statement (SQLSTATE 42613).

EXTENTSIZE *number-of-pages*
Specifies the number of PAGESIZE pages that will be written to a container before skipping to the next container. The extent size value can also be specified as an integer value followed by K (for kilobytes) or M (for megabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the value for the extent size. The database manager cycles repeatedly through the containers as data is stored.

The default value is provided by the DFT_EXTENT_SZ database configuration parameter, which has a valid range of 2-256 pages.

PREFETCHSIZE
Specifies to read in data needed by a query prior to it being referenced by the query, so that the query need not wait for I/O to be performed.

The default value is provided by the DFT_PREFETCH_SZ database configuration parameter.

AUTOMATIC
Specifies that the prefetch size of a table space is to be updated automatically; that is, the prefetch size will be managed by DB2, using the following formula:

```
Prefetch size =
 (number of containers) *
 (number of physical disks per container) *
 (extent size)
```

The number of physical disks per container defaults to 1, unless a value is specified through the DB2_PARALLEL_IO registry variable.

DB2 will update the prefetch size automatically whenever the number of containers in a table space changes (following successful execution of an ALTER TABLESPACE statement that adds or drops one or more containers). The prefetch size is updated at database start-up.

number-of-pages
Specifies the number of PAGESIZE pages that will be read from the table space when data prefetching is being performed. The prefetch size value can also be specified as an integer value followed by K (for kilobytes), M (for megabytes), or G (for gigabytes). If specified in this way, the floor of the number of bytes divided by the page size is used to determine the number of pages value for prefetch size.

BUFFERPOOL *bufferpool-name*
The name of the buffer pool used for tables in this table space. The buffer pool must exist (SQLSTATE 42704). If not specified, the default buffer pool (IBMDEFAULTBP) is used. The page size of the buffer pool must match the page size specified (or defaulted) for the table space (SQLSTATE

428CB). The database partition group of the table space must be defined for the buffer pool (SQLSTATE 42735).

OVERHEAD *number-of-milliseconds*
Specifies the I/O controller overhead and disk seek and latency time. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

For a database that was created in Version 9 or later, the default I/O controller overhead and disk seek and latency time is 7.5 milliseconds. For a database that was migrated from a previous version of DB2 to Version 9 or later, the default is 12.67 milliseconds.

FILE SYSTEM CACHING or **NO FILE SYSTEM CACHING**
Specifies whether or not I/O operations are to be cached at the file system level. The default is FILE SYSTEM CACHING.

FILE SYSTEM CACHING
Specifies that all I/O operations in the target table space are to be cached at the file system level.

NO FILE SYSTEM CACHING
Specifies that all I/O operations are to bypass the file system-level cache.

TRANSFERRATE *number-of-milliseconds*
Specifies the time to read one page into memory. This value is used to determine the cost of I/O during query optimization. The value of *number-of-milliseconds* is any numeric literal (integer, decimal, or floating point). If this value is not the same for all containers, the number should be the average for all containers that belong to the table space.

For a database that was created in Version 9 or later, the default time to read one page into memory is 0.06 milliseconds. For a database that was migrated from a previous version of DB2 to Version 9 or later, the default is 0.18 milliseconds.

DROPPED TABLE RECOVERY
Indicates whether dropped tables in the specified table space can be recovered using the RECOVER DROPPED TABLE option of the ROLLFORWARD DATABASE command. This clause can only be specified for a regular table space (SQLSTATE 42613).

ON
Specifies that dropped tables can be recovered. This has been the default since Version 8.

OFF
Specifies that dropped tables cannot be recovered. This is the default in Version 7.

Rules:

- If automatic storage is not defined for the database, an error is returned (SQLSTATE 55060).
- The INITIALSIZE clause cannot be specified with the MANAGED BY SYSTEM or MANAGED BY DATABASE clause (SQLSTATE 42601).
- The AUTORESIZE, INCREASESIZE, or MAXSIZE clause cannot be specified with the MANAGED BY SYSTEM clause (SQLSTATE 42601).

- The AUTORESIZE, INITIALSIZE, INCREASESIZE, or MAXSIZE clause cannot be specified for the creation of a temporary automatic storage table space (SQLSTATE 42601).
- The INCREASESIZE or MAXSIZE clause cannot be specified if the tables space is not auto-resizable (SQLSTATE 42601).
- AUTORESIZE cannot be enabled for DMS table spaces that are defined to use raw device containers (SQLSTATE 42601).
- A table space must initially be large enough to hold five extents (SQLSTATE 57011).
- The maximum size of a table space must be larger than its initial size (SQLSTATE 560B0).
- Container operations (ADD, EXTEND, RESIZE, REDUCE, DROP, or BEGIN STRIPE SET) cannot be performed on automatic storage table spaces, because the database manager is controlling the space management of such table spaces (SQLSTATE 42858).
- Each container definition requires 53 bytes plus the number of bytes necessary to store the container name. The combined length of all container names for the table space cannot exceed 20 480 bytes (SQLSTATE 54034).
- For a partitioned database, if more than one database partition resides on the same physical node, the same device or path cannot be specified for more than one database partition (SQLSTATE 42730). In this environment, either specify a unique *container-string* for each database partition, or use a relative path name.

Notes:

- Choosing between a database-managed space or a system-managed space for a table space is a fundamental choice involving trade-offs.
- When more than one TEMPORARY table space exists in the database, they are used in round-robin fashion to balance their usage.
- You can specify a database partition expression for container string syntax when creating either SMS or DMS containers. You would typically specify the database partition expression when using multiple logical database partitions in the partitioned database system. This ensures that container names are unique across database partition servers. When the expression is specified, the database partition number is part of the container name or, if additional arguments are specified, the result of the argument is part of the container name.

  You use the argument " $N" ([blank]$N) to indicate a database partition expression. A database partition expression can be used anywhere in the container name, and multiple database partition expressions can be specified. Terminate the database partition expression with a space character; whatever follows the space is appended to the container name after the database partition expression is evaluated. If there is no space character in the container name after the database partition expression, it is assumed that the rest of the string is part of the expression. The argument can only be used in one of the following forms.

Table 144. Arguments for Creating Containers. Operators are evaluated from left to right. The database partition number in the examples is assumed to be 5.

Syntax	Example	Value
[blank]$N	" $N"	5
[blank]$N+[number]	" $N+1011"	1016
[blank]$N%[number]	" $N%3" [a]	2
[blank]$N+[number]%[number]	" $N+12%13"	4

Table 144. Arguments for Creating Containers (continued). Operators are evaluated from left to right. The database partition number in the examples is assumed to be 5.

Syntax	Example	Value
[blank]$N%[number]+[number]	" $N%3+20"	22
[a] % represents the modulus operator.		

For example:

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
  (device '/dev/rcont $N' 20000)

On a two database partition system, the following containers
 would be created:
  /dev/rcont0  - on DATABASE PARTITION 0
  /dev/rcont1  - on DATABASE PARTITION 1


CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
  (file '/DB2/containers/TS2/container $N+100' 10000)

On a four database partition system, the following containers
 would be created:
  /DB2/containers/TS2/container100  - on DATABASE PARTITION 0
  /DB2/containers/TS2/container101  - on DATABASE PARTITION 1
  /DB2/containers/TS2/container102  - on DATABASE PARTITION 2
  /DB2/containers/TS2/container103  - on DATABASE PARTITION 3


CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
  ('/TS3/cont $N%2','/TS3/cont $N%2+2')

On a two database partition system, the following containers
 would be created:
  /TS3/cont0  - On DATABASE PARTITION 0
  /TS3/cont2  - On DATABASE PARTITION 0
  /TS3/cont1  - On DATABASE PARTITION 1
  /TS3/cont3  - On DATABASE PARTITION 1


If database partition = 5, the containers:
  '/dbdir/node $N /cont1'
  '/ $N+1000 /file1'
  ' $N%10 /container'
  '/dir/ $N%5+2000 /dmscont'

are created as:
  '/dbdir/node5/cont1'
  '/1005/file1'
  '5/container'
  '/dir/2000/dmscont'
```

- An automatic storage table space is created as either an SMS table space or a DMS table space. DMS is chosen for large and regular table spaces, and SMS is chosen for temporary table spaces. Note that this behavior cannot be depended upon, because it might change in a future release. When DMS is chosen and the type of table space is not specified, the default behavior is to create a large table space.
- The creation of an automatic storage table space does not include container definitions. The database manager automatically determines the location and size, if applicable, of the containers on the basis of the storage paths that are associated with the database. The database manager will attempt to grow large and regular table spaces, as necessary, provided that the maximum size has not been reached. This might involve extending existing containers or adding

containers to a new stripe set. Every time that the database is activated, the database manager automatically reconfigures the number and location of the containers for temporary table spaces that are not in an abnormal state.

- A large or regular automatic storage table space will not use new storage paths (see the description of the ALTER DATABASE statement) until there is no more space in one of the existing storage paths that the table space is using. Temporary automatic storage table spaces can only use the new storage paths once the database has been deactivated and then reactivated.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM
    - NODES can be specified in place of DBPARTITIONNUMS
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
    - LONG can be specified in place of LARGE

Examples:

Example 1: Create a regular DMS table space on a UNIX system using three devices of 10 000 4K pages each. Specify their I/O characteristics.

```
CREATE TABLESPACE PAYROLL
  MANAGED BY DATABASE
  USING (DEVICE'/dev/rhdisk6' 10000,
    DEVICE '/dev/rhdisk7' 10000,
    DEVICE '/dev/rhdisk8' 10000)
  OVERHEAD 12.67
  TRANSFERRATE 0.18
```

Example 2: Create a regular SMS table space on Windows using three directories on three separate drives, with a 64-page extent size, and a 32-page prefetch size.

```
CREATE TABLESPACE ACCOUNTING
  MANAGED BY SYSTEM
  USING ('d:\acc_tbsp', 'e:\acc_tbsp', 'f:\acc_tbsp')
  EXTENTSIZE 64
  PREFETCHSIZE 32
```

Example 3: Create a temporary DMS table space on a UNIX system using two files of 50 000 pages each, and a 256-page extent size.

```
CREATE TEMPORARY TABLESPACE TEMPSPACE2
  MANAGED BY DATABASE
  USING (FILE 'dbtmp/tempspace2.f1' 50000,
    FILE 'dbtmp/tempspace2.f2' 50000)
  EXTENTSIZE 256
```

Example 4: Create a DMS table space in database partition group ODDNODEGROUP (database partitions 1, 3, and 5) on a UNIX system. Use the device /dev/rhdisk0 for 10 000 4K pages on each database partition. Specify a database partition-specific device with 40 000 4K pages for each database partition.

```
CREATE TABLESPACE PLANS
  MANAGED BY DATABASE
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn1hd01' 40000)
  ON DBPARTITIONNUM (1)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn3hd03' 40000)
  ON DBPARTITIONNUM (3)
  USING (DEVICE '/dev/rhdisk0' 10000, DEVICE '/dev/rn5hd05' 40000)
  ON DBPARTITIONNUM (5)
```

Example 5: Create a large automatic storage table space named DATATS, allowing the system to make all decisions with respect to table space size and growth.

```
CREATE TABLESPACE DATATS
```

or

```
CREATE TABLESPACE DATATS
  MANAGED BY AUTOMATIC STORAGE
```

Example 6: Create a temporary automatic storage table space named TEMPDATA.

```
CREATE TEMPORARY TABLESPACE TEMPDATA
```

or

```
CREATE TEMPORARY TABLESPACE TEMPDATA
  MANAGED BY AUTOMATIC STORAGE
```

Example 7: Create a regular automatic storage table space named USERSPACE3 with an initial size of 100 megabytes and a maximum size of 1 gigabyte.

```
CREATE TABLESPACE USERSPACE3
  INITIALSIZE 100 M
  MAXSIZE 1 G
```

Example 8: Create a large automatic storage table space named LARGEDATA with a growth rate of 10 percent (that is, its total size increases by 10 percent each time that it is automatically resized) and a maximum size of 512 megabytes. Instead of specifying the INITIALSIZE clause, let the database manager determine an appropriate initial size for the table space.

```
CREATE LARGE TABLESPACE LARGEDATA
  INCREASESIZE 10 PERCENT
  MAXSIZE 512 M
```

Example 9: Create a regular DMS table space named USERSPACE4 with two file containers (each container being 1 megabyte in size), a growth rate of 2 megabytes, and a maximum size of 100 megabytes.

```
CREATE TABLESPACE USERSPACE4
  MANAGED BY DATABASE USING (FILE '/db2/file1' 1 M, FILE '/db2/file2' 1 M)
  AUTORESIZE YES
  INCREASESIZE 2 M
  MAXSIZE 100 M
```

Example 10: Create regular DMS table spaces, using RAW devices on a Windows operating system.

- To specify entire physical drives, use the \\.*physical-drive* format:

  ```
  CREATE TABLESPACE TS1
    MANAGED BY DATABASE USING (DEVICE '\\.\PhysicalDrive5' 10000,
      DEVICE '\\.\PhysicalDrive6' 10000)
  ```

- To specify logical partitions by using drive letters:

  ```
  CREATE TABLESPACE TS2
    MANAGED BY DATABASE USING (DEVICE '\\.\G:' 10000,
      DEVICE '\\.\H:' 10000)
  ```

- To specify logical partitions by using volume global unique identifiers (GUIDs), use the db2listvolumes utility to retrieve the volume GUID for each local partition, then copy the GUID for the logical partition that you want into the table space container clause:

  ```
  CREATE TABLESPACE TS3
    MANAGED BY DATABASE USING (
      DEVICE '\\?\Volume{2ca6a0c1-8542-11d8-9734-00096b5322d2}\' 20000M)
  ```

You might prefer to use volume GUIDs over the drive letter format if you have more partitions than available drive letters on the machine.

- To specify logical partitions by using junction points (or volume mount points), mount the RAW partition to another NTFS-formatted volume as a junction point, then specify the path to the junction point on the NTFS volume as the container path. For example:

```
CREATE TABLESPACE TS4
  MANAGED BY DATABASE USING (DEVICE 'C:\JUNCTION\DISK_1' 10000,
    DEVICE 'C:\JUNCTION\DISK_2' 10000)
```

  DB2 first queries the partition to see whether there is a file system on it; if yes, the partition is not treated as a RAW device, and DB2 performs normal file system I/O operations on the partition.

Related concepts:

- "Automatic resizing of table spaces" in *Administration Guide: Implementation*
- "Automatic storage databases" in *Administration Guide: Implementation*

Related tasks:

- "Attaching a direct disk access device" in *Administration Guide: Implementation*

Related reference:

- "SQL and XQuery limits" in *SQL Reference, Volume 1*
- "ALTER TABLESPACE " on page 898

Related samples:

- "tbtemp.sqc -- How to use a declared temporary table (C)"
- "TbTemp.java -- How to use Declared Temporary Table (JDBC)"

CREATE VIEW

The CREATE VIEW statement defines a view on one or more tables, views or nicknames.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- For each table, view, or nickname identified in any fullselect:
  – CONTROL privilege on that table or view, or
  – SELECT privilege on that table or view

  and at least one of the following:
  – IMPLICIT_SCHEMA authority on the database, if the implicit or explicit schema name of the view does not exist

  – CREATEIN privilege on the schema, if the schema name of the view refers to an existing schema

  If creating a subview, the authorization ID of the statement must:

  – Be the same as the definer of the root table of the table hierarchy
  – Have SELECT WITH GRANT privilege on the underlying table of the subview or the superview
  – Must not have SELECT privilege granted to any user other than the view definer

- SYSADM or DBADM authority

Group privileges are not considered for any table or view specified in the CREATE VIEW statement.

Privileges are not considered when defining a view on a federated database nickname. Authorization requirements of the data source for the table or view referenced by the nickname are applied when the query is processed. The authorization ID of the statement can be mapped to a different remote authorization ID.

If a view definer can only create the view because the definer has SYSADM authority, the definer is granted explicit DBADM authority for the purpose of creating the view.

Syntax:

```
>>──CREATE──VIEW──view-name──┬───────────────────────────────────────────────────────┬──────>
                             │    ┌─,───────────┐                                    │
                             │    ▼             │                                    │
                             ├─(───column-name──┴─)──────────────────────────────────┤
                             └─OF──type-name──┬─┤ root-view-definition ├─┬───────────┘
                                              └─┤ subview-definition ├───┘

>─AS──┬─────────────────────────────────────────┬──fullselect──●──────────────────────────>
      │        ┌─,──────────────────────┐       │
      │        ▼                        │       │
      └─WITH────common-table-expression─┴───────┘

>──┬──────────────────────────────────────────┬──●──┬─WITH NO ROW MOVEMENT─┬──●─────────><
   │        ┌─CASCADED─┐                      │     └─WITH ROW MOVEMENT────┘
   └─WITH───┼──────────┼───CHECK OPTION───────┘
            └─LOCAL────┘
```

root-view-definition:

```
├──MODE DB2SQL──(──┤ oid-column ├──┬─────────────────────────┬──)──────────────────────┤
                                   └─,──┤ with-options ├─────┘
```

subview-definition:

```
├──MODE DB2SQL──┤ under-clause ├──┬────────────────────────────┬──┬────────┬───────────┤
                                  └─(──┤ with-options ├──)─────┘  └─EXTEND─┘
```

oid-column:

```
├──REF IS──oid-column-name──USER GENERATED──┬───────────┬──────────────────┤
                                            └─UNCHECKED─┘
```

with-options:

```
   ┌─,────────────────────────────────────────────────────────────┐
   │                          ┌─,──────────────────────────────┐  │
   ▼                          ▼                                │  │
├────column-name──WITH OPTIONS────┬─SCOPE──┬─typed-table-name─┬─┬─┴──┴──────┤
                                  │        └─typed-view-name──┘ │
                                  └─READ ONLY───────────────────┘
```

under-clause:

```
├──UNDER──superview-name──INHERIT SELECT PRIVILEGES──────────────────────┤
```

Description:

view-name
: Names the view. The name, including the implicit or explicit qualifier, must not identify a table, view, nickname or alias described in the catalog. The qualifier must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42939).

  The name can be the same as the name of an inoperative view (see "Inoperative views" on page 1042). In this case the new view specified in the CREATE VIEW statement will replace the inoperative view. The user will get a warning (SQLSTATE 01595) when an inoperative view is replaced. No warning is returned if the application was bound with the bind option SQLWARN set to NO.

column-name
: Names the columns in the view. If a list of column names is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If a list of column names is not specified, the columns of the view inherit the names of the columns of the result table of the fullselect.

  A list of column names must be specified if the result table of the fullselect has duplicate column names or an unnamed column (SQLSTATE 42908). An unnamed column is a column derived from a constant, function, expression, or set operation that is not named using the AS clause of the select list.

OF *type-name*
: Specifies that the columns of the view are based on the attributes of the structured type identified by *type-name*. If *type-name* is specified without a schema name, the type name is resolved by searching the schemas on the SQL path (defined by the FUNCPATH preprocessing option for static SQL and by the CURRENT PATH register for dynamic SQL). The type name must be the name of an existing user-defined type (SQLSTATE 42704) and it must be a structured type that is instantiable (SQLSTATE 428DP).

MODE DB2SQL
: This clause is used to specify the mode of the typed view. This is the only valid mode currently supported.

UNDER *superview-name*
Indicates that the view is a subview of *superview-name*. The superview must be an existing view (SQLSTATE 42704) and the view must be defined using a structured type that is the immediate supertype of *type-name* (SQLSTATE 428DB). The schema name of *view-name* and *superview-name* must be the same (SQLSTATE 428DQ). The view identified by *superview-name* must not have any existing subview already defined using *type-name* (SQLSTATE 42742).

The columns of the view include the object identifier column of the superview with its type modified to be REF(*type-name*), followed by columns based on the attributes of *type-name* (remember that the type includes the attributes of its supertype).

INHERIT SELECT PRIVILEGES
Any user or group holding a SELECT privilege on the superview will be granted an equivalent privilege on the newly created subview. The subview definer is considered to be the grantor of this privilege.

OID-column
Defines the object identifier column for the typed view.

REF IS *OID-column-name* **USER GENERATED**
Specifies that an object identifier (OID) column is defined in the view as the first column. An OID is required for the root view of a view hierarchy (SQLSTATE 428DX). The view must be a typed view (the OF clause must be present) that is not a subview (SQLSTATE 42613). The name for the column is defined as *OID-column-name* and cannot be the same as the name of any attribute of the structured type *type-name* (SQLSTATE 42711). The first column specified in *fullselect* must be of type REF(*type-name*) (you may need to cast it so that it has the appropriate type). If UNCHECKED is not specified, it must be based on a not nullable column on which uniqueness is enforced through an index (primary key, unique constraint, unique index, or OID-column). This column will be referred to as the *object identifier column* or *OID column*. The keywords USER GENERATED indicate that the initial value for the OID column must be provided by the user when inserting a row. Once a row is inserted, the OID column cannot be updated (SQLSTATE 42808).

UNCHECKED
Defines the object identifier column of the typed view definition to assume uniqueness even though the system can not prove this uniqueness. This is intended for use with tables or views that are being defined into a typed view hierarchy where the user knows that the data conforms to this uniqueness rule but it does not comply with the rules that allow the system to prove uniqueness. UNCHECKED option is mandatory for view hierarchies that range over multiple hierarchies or legacy tables or views By specifying UNCHECKED, the user takes responsibility for ensuring that each row of the view has a unique OID. If the user fails to ensure this property, and a view contains duplicate OID values, then a path-expression or DEREF operator involving one of the non-unique OID values may result in an error (SQLSTATE 21000).

with-options
Defines additional options that apply to columns of a typed view.

column-name **WITH OPTIONS**
Specifies the name of the column for which additional options are specified. The *column-name* must correspond to the name of an attribute defined in (not inherited by) the *type-name* of the view. The column must

be a reference type (SQLSTATE 42842). It cannot correspond to a column that also exists in the superview (SQLSTATE 428DJ). A column name can only appear in one WITH OPTIONS SCOPE clause in the statement (SQLSTATE 42613).

SCOPE
Identifies the scope of the reference type column. A scope must be specified for any column that is intended to be used as the left operand of a dereference operator or as the argument of the DEREF function.

Specifying the scope for a reference type column may be deferred to a subsequent ALTER VIEW statement (if the scope is not inherited) to allow the target table or view to be defined, usually in the case of mutually referencing views and tables. If no scope is specified for a reference type column of the view and the underlying table or view column was scoped, then the underlying column's scope is inherited by the reference type column. The column remains unscoped if the underlying table or view column did not have a scope. See 1041 for more information about scope and reference type columns.

typed-table-name
The name of a typed table. The table must already exist or be the same as the name of the table being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-table-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-table-name*.

typed-view-name
The name of a typed view. The view must already exist or be the same as the name of the view being created (SQLSTATE 42704). The data type of *column-name* must be REF(*S*), where *S* is the type of *typed-view-name* (SQLSTATE 428DM). No checking is done of any existing values in *column-name* to ensure that the values actually reference existing rows in *typed-view-name*.

READ ONLY
Identifies the column as a read-only column. This option is used to force a column to be read-only so that subview definitions can specify an expression for the same column that is implicitly read-only.

AS
Identifies the view definition.

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows. A common table expression cannot be specified when defining a typed view.

fullselect
Defines the view. At any time, the view consists of the rows that would result if the SELECT statement were executed. The fullselect must not reference host variables, parameter markers, or declared temporary tables. However, a parameterized view can be created as an SQL table function.

The fullselect cannot include an SQL data change statement in the FROM clause (SQLSTATE 428FL).

For Typed Views and Subviews: The *fullselect* must conform to the following rules otherwise an error is returned (SQLSTATE 428EA unless otherwise specified).

- The fullselect must not include references to the DBPARTITIONNUM or HASHEDVALUE functions, non-deterministic functions, or functions defined to have external action.
- The body of the view must consist of a single subselect, or a UNION ALL of two or more subselects. Let each of the subselects participating directly in the view body be called a *branch* of the view. A view may have one or more branches.
- The FROM-clause of each branch must consist of a single table or view (not necessarily typed), called the *underlying* table or view of that branch.
- The underlying table or view of each branch must be in a separate hierarchy (that is, a view cannot have multiple branches with their underlying tables or views in the same hierarchy).
- None of the branches of a typed view definition may specify GROUP BY or HAVING.
- If the view body contains UNION ALL, the root view in the hierarchy must specify the UNCHECKED option for its OID column.

For a hierarchy of views and subviews: Let BR1 and BR2 be any branches that appear in the definitions of views in the hierarchy. Let T1 be the underlying table or view of BR1, and let T2 be the underlying table or view of BR2. Then:

- If T1 and T2 are not in the same hierarchy, then the root view in the view hierarchy must specify the UNCHECKED option for its OID column.
- If T1 and T2 are in the same hierarchy, then BR1 and BR2 must contain predicates or ONLY-clauses that are sufficient to guarantee that their row-sets are disjoint.

For typed subviews defined using EXTEND AS: For every branch in the body of the subview:

- The underlying table of each branch must be a (not necessarily proper) subtable of some underlying table of the immediate superview.
- The expressions in the SELECT list must be assignable to the non-inherited columns of the subview (SQLSTATE 42854).

For typed subviews defined using AS without EXTEND:

- For every branch in the body of the subview, the expressions in the SELECT-list must be assignable to the declared types of the inherited and non-inherited columns of the subview (SQLSTATE 42854).
- The OID-expression of each branch over a given hierarchy in the subview must be equivalent (except for casting) to the OID-expression in the branch over the same hierarchy in the root view.
- The expression for a column not defined (implicitly or explicitly) as READ ONLY in a superview must be equivalent in all branches over the same underlying hierarchy in its subviews.

WITH CHECK OPTION

Specifies the constraint that every row that is inserted or updated through the view must conform to the definition of the view. A row that does not conform to the definition of the view is a row that does not satisfy the search conditions of the view.

WITH CHECK OPTION must not be specified if any of the following conditions is true:

- The view is read-only (SQLSTATE 42813). If WITH CHECK OPTION is specified for an updatable view that does not allow inserts, the constraint applies to updates only.
- The view references the DBPARTITIONNUM or HASHEDVALUE function, a non-deterministic function, or a function with external action (SQLSTATE 42997).
- A nickname is the update target of the view.
- A view that has an INSTEAD OF trigger defined on it is the update target of the view (SQLSTATE 428FQ).

If WITH CHECK OPTION is omitted, the definition of the view is not used in the checking of any insert or update operations that use the view. Some checking might still occur during insert or update operations if the view is directly or indirectly dependent on another view that includes WITH CHECK OPTION. Because the definition of the view is not used, rows might be inserted or updated through the view that do not conform to the definition of the view.

CASCADED
The WITH CASCADED CHECK OPTION constraint on a view *V* means that *V* inherits the search conditions as constraints from any updatable view on which *V* is dependent. Furthermore, every updatable view that is dependent on *V* is also subject to these constraints. Thus, the search conditions of *V* and each view on which *V* is dependent are ANDed together to form a constraint that is applied for an insert or update of *V* or of any view dependent on *V*.

LOCAL
The WITH LOCAL CHECK OPTION constraint on a view *V* means the search condition of *V* is applied as a constraint for an insert or update of *V* or of any view that is dependent on *V*.

The difference between CASCADED and LOCAL is shown in the following example. Consider the following updatable views (substituting for Y from column headings of the table that follows):

```
V1 defined on table T
V2 defined on V1 WITH Y CHECK OPTION
V3 defined on V2
V4 defined on V3 WITH Y CHECK OPTION
V5 defined on V4
```

The following table shows the search conditions against which inserted or updated rows are checked:

	Y is LOCAL	**Y is CASCADED**
V1 checked against:	no view	no view
V2 checked against:	V2	V2, V1
V3 checked against:	V2	V2, V1
V4 checked against:	V2, V4	V4, V3, V2, V1
V5 checked against:	V2, V4	V4, V3, V2, V1

Consider the following updatable view which shows the impact of the WITH CHECK OPTION using the default CASCADED option:

```
CREATE VIEW V1 AS SELECT COL1 FROM T1 WHERE COL1 > 10

CREATE VIEW V2 AS SELECT COL1 FROM V1 WITH CHECK OPTION

CREATE VIEW V3 AS SELECT COL1 FROM V2 WHERE COL1 < 100
```

The following INSERT statement using *V1* will succeed because *V1* does not have a WITH CHECK OPTION and *V1* is not dependent on any other view that has a WITH CHECK OPTION.

```
INSERT INTO V1 VALUES(5)
```

The following INSERT statement using *V2* will result in an error because *V2* has a WITH CHECK OPTION and the insert would produce a row that did not conform to the definition of *V2*.

```
INSERT INTO V2 VALUES(5)
```

The following INSERT statement using *V3* will result in an error even though it does not have WITH CHECK OPTION because *V3* is dependent on *V2* which does have a WITH CHECK OPTION (SQLSTATE 44000).

```
INSERT INTO V3 VALUES(5)
```

The following INSERT statement using *V3* will succeed even though it does not conform to the definition of *V3* (*V3* does not have a WITH CHECK OPTION); it does conform to the definition of *V2* which does have a WITH CHECK OPTION.

```
INSERT INTO V3 VALUES(200)
```

WITH NO ROW MOVEMENT or **WITH ROW MOVEMENT**
Specifies the action to take for an updatable UNION ALL view when a row is updated in a way that violates a check constraint on the underlyig table. The default is WITH NO ROW MOVEMENT.

WITH NO ROW MOVEMENT
Specifies that an error (SQLSTATE 23513) is to be returned if a row is updated in a way that violates a check constraint on the underlying table.

WITH ROW MOVEMENT
Specifies that an updated row is to be moved to the appropriate underlying table, even if it violates a check constraint on that table.

Row movement involves deletion of the rows that violate the check constraint, and insertion of those rows back into the view. The WITH ROW MOVEMENT clause can only be specified for UNION ALL views whose columns are all updatable (SQLSTATE 429BJ). If a row is inserted (perhaps after trigger activation) into the same underlying table from which it was deleted, an error is returned (SQLSTATE 23524). A view defined using the WITH ROW MOVEMENT clause must not contain nested UNION ALL operations, except in the outermost fullselect (SQLSTATE 429BJ).

Notes:

- Creating a view with a schema name that does not already exist will result in the implicit creation of that schema provided the authorization ID of the statement has IMPLICIT_SCHEMA authority. The schema owner is SYSIBM. The CREATEIN privilege on the schema is granted to PUBLIC.
- View columns inherit the NOT NULL WITH DEFAULT attribute from the base table or view except when columns are derived from an expression. When a row

is inserted or updated into an updatable view, it is checked against the constraints (primary key, referential integrity, and check) if any are defined on the base table.

- A new view cannot be created if it uses an inoperative view in its definition. (SQLSTATE 51024).
- This statement does not support declared temporary tables (SQLSTATE 42995).
- ***Deletable views:*** A view is *deletable* if an INSTEAD OF trigger for the delete operation has been defined for the view, or if all of the following are true:
  - each FROM clause of the outer fullselect identifies only one base table (with no OUTER clause), deletable view (with no OUTER clause), deletable nested table expression, or deletable common table expression (cannot identify a nickname)
  - the outer fullselect does not include a VALUES clause
  - the outer fullselect does not include a GROUP BY clause or HAVING clause
  - the outer fullselect does not include column functions in the select list
  - the outer fullselect does not include SET operations (UNION, EXCEPT or INTERSECT) with the exception of UNION ALL
  - the base tables in the operands of a UNION ALL must not be the same table and each operand must be deletable
  - the select list of the outer fullselect does not include DISTINCT
- ***Updatable views:*** A column of a view is *updatable* if an INSTEAD OF trigger for the update operation has been defined for the view, or if all of the following are true:
  - the view is deletable (independent of an INSTEAD OF trigger for delete), the column resolves to a column of a base table (not using a dereference operation), and the READ ONLY option is not specified
  - all the corresponding columns of the operands of a UNION ALL have exactly matching data types (including length or precision and scale) and matching default values if the fullselect of the view includes a UNION ALL

  A view is updatable if *any* column of the view is updatable.
- ***Insertable views:***
  - A view is insertable if an INSTEAD OF trigger for the insert operation has been defined for the view, or at least one column of the view is updatable (independent of an INSTEAD OF trigger for update), and the fullselect of the view does not include UNION ALL.
  - A given row can be inserted into a view (including a UNION ALL) if, and only if, it fulfills the check constraints of exactly one of the underlying base tables.
  - To insert into a view that includes non-updatable columns, those columns must be omitted from the column list.
- ***Read-only views:*** A view is *read-only* if it is *not* deletable, updatable, or insertable.

  The READONLY column in the SYSCAT.VIEWS catalog view indicates if a view is read-only without considering INSTEAD OF triggers.
- Common table expressions and nested table expressions follow the same set of rules for determining whether they are deletable, updatable, insertable, or read-only.
- ***Inoperative views:*** An *inoperative view* is a view that is no longer available for SQL statements. A view becomes inoperative if:
  - A privilege, upon which the view definition is dependent, is revoked.

  – An object such as a table, nickname, alias or function, upon which the view definition is dependent, is dropped.
  – A view, upon which the view definition is dependent, becomes inoperative.
  – A view that is the superview of the view definition (the subview) becomes inoperative.

  In practical terms, an inoperative view is one in which the view definition has been unintentionally dropped. For example, when an alias is dropped, any view defined using that alias is made inoperative. All dependent views also become inoperative and packages dependent on the view are no longer valid.

  Until the inoperative view is explicitly recreated or dropped, a statement using that inoperative view cannot be compiled (SQLSTATE 51024) with the exception of the CREATE ALIAS, CREATE VIEW, DROP VIEW, and COMMENT ON TABLE statements. Until the inoperative view has been explicitly dropped, its qualified name cannot be used to create another table or alias (SQLSTATE 42710).

  An inoperative view may be recreated by issuing a CREATE VIEW statement using the definition text of the inoperative view. This view definition text is stored in the TEXT column of the SYSCAT.VIEWS catalog. When recreating an inoperative view, it is necessary to explicitly grant any privileges required on that view by others, due to the fact that all authorization records on a view are deleted if the view is marked inoperative. Note that there is no need to explicitly drop the inoperative view in order to recreate it. Issuing a CREATE VIEW statement with the same *view-name* as an inoperative view will cause that inoperative view to be replaced, and the CREATE VIEW statement will return a warning (SQLSTATE 01595).

  Inoperative views are indicated by an X in the VALID column of the SYSCAT.VIEWS catalog view and an X in the STATUS column of the SYSCAT.TABLES catalog view.

- ***Privileges:***

  The definer of a view always receives the SELECT privilege on the view as well as the right to drop the view. The definer of a view will get CONTROL privilege on the view only if the definer has CONTROL privilege on every base table, view, or nickname identified in the fullselect, or if the definer has SYSADM or DBADM authority.

  The definer of the view is granted INSERT, UPDATE, column level UPDATE or DELETE privileges on the view if the view is not read-only and the definer has the corresponding privileges on the underlying objects.

  For a view defined WITH ROW MOVEMENT, the definer acquires the UPDATE privilege on the view only if the definer has the UPDATE privilege on all columns of the view, as well as INSERT and DELETE privileges on all underlying tables or views.

  The definer of a view only acquires privileges if the privileges from which they are derived exist at the time the view is created. The definer must have these privileges either directly or because PUBLIC has these privilege. Privileges are not considered when defining a view on a federated server nickname. However, when using a view on a nickname, the user's authorization ID must have valid select privileges on the table or view that the nickname references at the data source. Otherwise, an error is returned. Privileges held by groups of which the view definer is a member, are not considered.

  When a subview is created, the SELECT privileges held on the immediate superview are automatically granted on the subview.

- ***Scope and REF columns:***

When selecting a reference type column in the fullselect of a view definition, consider the target type and scope that is required.

- If the required target type and scope is the same as the underlying table or view, the column can simply be selected.
- If the scope needs to be changed, use the WITH OPTIONS SCOPE clause to define the required scope table or view.
- If the target type of the reference needs to be changed, the column must be cast first to the representation type of the reference and then to the new reference type. The scope in this case can be specified in the cast to the reference type or using the WITH OPTIONS SCOPE clause. For example, assume you select column Y defined as REF(TYP1) SCOPE TAB1. You want this to be defined as REF(VTYP1) SCOPE VIEW1. The select list item would be as follows:

```
CAST(CAST(Y AS VARCHAR(16) FOR BIT DATA) AS REF(VTYP1) SCOPE VIEW1)
```

- ***Identity columns:*** A column of a view is considered an identity column, if the element of the corresponding column in the fullselect of the view definition is the name of an identity column of a table, or the name of a column of a view which directly or indirectly maps to the name of an identity column of a base table.

  In all other cases, the columns of a view will not get the identity property. For example:

  - the select-list of the view definition includes multiple instances of the name of an identity column (that is, selecting the same column more than once)
  - the view definition involves a join
  - a column in the view definition includes an expression that refers to an identity column
  - the view definition includes a UNION

  When inserting into a view for which the select list of the view definition directly or indirectly includes the name of an identity column of a base table, the same rules apply as if the INSERT statement directly referenced the identity column of the base table.

- ***Federated views:*** A federated view is a view that includes a reference to a nickname somewhere in the fullselect. The presence of such a nickname changes the authorization model used for the view when the view is subsequently referenced in a query.

  When the view is created, no privilege checking is done to determine whether the view definer has access to the underlying data source table or view of a nickname. Privilege checking of references to tables or views at the federated database are handled as usual, requiring the view definer to have at least SELECT privilege on such objects.

  When a federated view is subsequently referenced in a query, the nicknames result in queries against the data source, and the authorization ID that issued the query (or the remote authorization ID to which it maps) must have the necessary privileges to access the data source table or view. The authorization ID that issues the query referencing the federated view is not required to have any additional privileges on tables or views (non-federated) that exist at the federated server.

- ***ROW MOVEMENT, triggers and constraints:*** When a view that is defined using the WITH ROW MOVEMENT clause is updated, the sequence of trigger and constraints operations is as follows:

  1. BEFORE UPDATE triggers are activated for all rows being updated, including rows that will eventually be moved.

2. The update operation is processed.
3. Constraints are processed for all updated rows.
4. AFTER UPDATE triggers (both row-level and statement-level) are activated in creation order, for all rows that satisfy the constraints after the update operation. Because this is an UPDATE statement, all UPDATE statement-level triggers are activated for all underlying tables.
5. BEFORE DELETE triggers are activated for all rows that did not satisfy the constraints after the update operation (these are the rows that are to be moved).
6. The delete operation is processed.
7. Constraints are processed for all deleted rows.
8. AFTER DELETE triggers (both row-level and statement-level) are activated in creation order, for all deleted rows. Statement-level triggers are activated for only those tables that are involved in the delete operation.
9. BEFORE INSERT triggers are activated for all rows being inserted (that is, the rows being moved). The new transition tables for the BEFORE INSERT triggers contain the input data provided by the user.
10. The insert operation is processed.
11. Constraints are processed for all inserted rows.
12. AFTER INSERT triggers (both row-level and statement-level) are activated in creation order, for all inserted rows. Statement-level triggers are activated for only those tables that are involved in the insert operation.

- ***Nested UNION ALL views:*** A view defined with UNION ALL and based, either directly or indirectly, on a view that is also defined with UNION ALL cannot be updated if either view is defined using the WITH ROW MOVEMENT clause (SQLSTATE 429BK).
- ***Compatibilities:***
  – For compatibility with previous versions of DB2:
    - The FEDERATED keyword can be specified between the keywords CREATE and VIEW. The FEDERATED keyword is ignored, however, because a warning is no longer returned if federated objects are used in the view definition.

Examples:

Example 1: Create a view named MA_PROJ upon the PROJECT table that contains only those rows with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE VIEW MA_PROJ  AS SELECT *
  FROM PROJECT
   WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 2: Create a view as in example 1, but select only the columns for project number (PROJNO), project name (PROJNAME) and employee in charge of the project (RESPEMP).

```
CREATE VIEW MA_PROJ
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 3: Create a view as in example 2, but, in the view, call the column for the employee in charge of the project IN_CHARGE.

```
CREATE VIEW MA_PROJ
  (PROJNO, PROJNAME, IN_CHARGE)
  AS SELECTPROJNO, PROJNAME, RESPEMP
  FROM PROJECT
  WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Note: Even though only one of the column names is being changed, the names of all three columns in the view must be listed in the parentheses that follow MA_PROJ.

Example 4: Create a view named PRJ_LEADER that contains the first four columns (PROJNO, PROJNAME, DEPTNO, RESPEMP) from the PROJECT table together with the last name (LASTNAME) of the person who is responsible for the project (RESPEMP). Obtain the name from the EMPLOYEE table by matching EMPNO in EMPLOYEE to RESPEMP in PROJECT.

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
```

Example 5: Create a view as in example 4, but in addition to the columns PROJNO, PROJNAME, DEPTNO, RESPEMP, and LASTNAME, show the total pay (SALARY + BONUS + COMM) of the employee who is responsible. Also select only those projects with mean staffing (PRSTAFF) greater than one.

```
CREATE VIEW PRJ_LEADER
(PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, TOTAL_PAY )
AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP, LASTNAME, SALARY+BONUS+COMM
  FROM PROJECT, EMPLOYEE
  WHERE RESPEMP = EMPNO
  AND PRSTAFF > 1
```

Specifying the column name list could be avoided by naming the expression SALARY+BONUS+COMM as TOTAL_PAY in the fullselect.

```
CREATE VIEW PRJ_LEADER
  AS SELECT PROJNO, PROJNAME, DEPTNO, RESPEMP,
             LASTNAME, SALARY+BONUS+COMM AS TOTAL_PAY
    FROM PROJECT, EMPLOYEE
    WHERE RESPEMP = EMPNO AND PRSTAFF > 1
```

Example 6: Given the set of tables and views shown in the following figure: User ZORPIE (who does not have either DBADM or SYSADM authority) has been

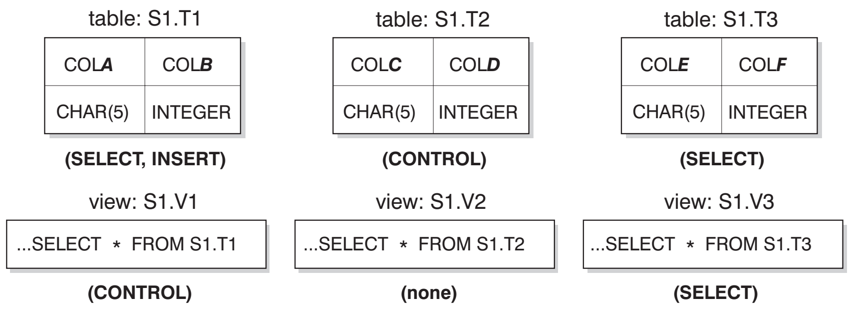


Figure 27. Tables and Views for Example 6

granted the privileges shown in brackets below each object:

1. ZORPIE will get CONTROL privilege on the view that she creates with:

```
CREATE VIEW VA AS SELECT * FROM S1.V1
```

because she has CONTROL on S1.V1. (CONTROL on S1.V1 must have been granted to ZORPIE by someone with DBADM or SYSADM authority.) It does not matter which, if any, privileges she has on the underlying base table.

2. ZORPIE will not be allowed to create the view:

```
CREATE VIEW VB AS SELECT * FROM S1.V2
```

   because she has neither CONTROL nor SELECT on S1.V2. It does not matter that she has CONTROL on the underlying base table (S1.T2).

3. ZORPIE will get CONTROL privilege on the view that she creates with:

```
CREATE VIEW VC (COLA, COLB, COLC, COLD)
  AS SELECT * FROM S1.V1, S1.T2
  WHERE COLA = COLC
```

   because the fullselect of ZORPIE.VC references view S1.V1 and table S1.T2 and she has CONTROL on both of these. Note that the view VC is read-only, so ZORPIE does not get INSERT, UPDATE or DELETE privileges.

4. ZORPIE will get SELECT privilege on the view that she creates with:

```
CREATE VIEW VD (COLA,COLB, COLE, COLF)
  AS SELECT * FROM S1.V1, S1.V3
  WHERE COLA = COLE
```

   because the fullselect of ZORPIE.VD references the two views S1.V1 and S1.V3, one on which she has only SELECT privilege, and one on which she has CONTROL privilege. She is given the lesser of the two privileges, SELECT, on ZORPIE.VD.

5. ZORPIE will get INSERT, UPDATE and DELETE privilege WITH GRANT OPTION and SELECT privilege on the view VE in the following view definition.

```
CREATE VIEW VE
   AS SELECT * FROM S1.V1
  WHERE COLA > ANY
         (SELECT COLE FROM S1.V3)
```

   ZORPIE's privileges on VE are determined primarily by her privileges on S1.V1. Since S1.V3 is only referenced in a subquery, she only needs SELECT privilege on S1.V3 to create the view VE. The definer of a view only gets CONTROL on the view if they have CONTROL on all objects referenced in the view definition. ZORPIE does not have CONTROL on S1.V3, consequently she does not get CONTROL on VE.

Related reference:

- "CREATE FUNCTION (SQL Scalar, Table, or Row) statement" in *SQL Reference, Volume 2*
- "SQL queries" on page 1210

DELETE

The DELETE statement deletes rows from a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect. Deleting a row from a nickname deletes the row from the data source object to which the nickname refers. Deleting a row from a view deletes the row from the table on which the view is based if no INSTEAD OF trigger is defined for the delete operation on this view. If such a trigger is defined, the trigger will be executed instead.

There are two forms of this statement:

- The *Searched* DELETE form is used to delete one or more rows (optionally determined by a search condition).
- The *Positioned* DELETE form is used to delete exactly one row (as determined by the current position of a cursor).

Invocation:

A DELETE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

To execute either form of this statement, the privileges held by the authorization ID of the statement must include at least one of the following:

- DELETE privilege on the table, view, or nickname from which rows are to be deleted
- CONTROL privilege on the table, view, or nickname from which rows are to be deleted
- SYSADM or DBADM authority

To execute a Searched DELETE statement, the privileges held by the authorization ID of the statement must also include at least one of the following for each table, view, or nickname referenced by a subquery:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of a DELETE statement includes a reference to a column of the table or view in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:

- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority

If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

Group privileges are not checked for static DELETE statements.

If the target of the delete operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax:

searched-delete:

```
>>──DELETE FROM──┬─table-name──────────────────┬──┬───────────────────────┬──────►
                 ├─view-name───────────────────┤  └─┤ correlation-clause ├─┘
                 ├─nickname────────────────────┤
                 ├─ONLY──(──┬─table-name─┬──)──┤
                 │          └─view-name──┘     │
                 └─(──fullselect──)────────────┘

►──┬───────────────────┬──┬─────────────────────┬───────────────────────────────►
   └─┤ include-columns ├─┘ └─┤ assignment-clause ├─┘

►──┬─────────────────────────┬──┬──────────────┬────────────────────────────────►◄
   └─WHERE──search-condition─┘  └─WITH──┬─RR─┬─┘
                                        ├─RS─┤
                                        ├─CS─┤
                                        └─UR─┘
```

include-columns:

```
                  ┌─,──────────────────────┐
                  ▼                        │
├──INCLUDE──(────column-name──data-type───┴──)───────────────────────────────┤
```

positioned-delete:

```
>>──DELETE FROM──┬─table-name──────────────────┬──┬───────────────────────┬──────►
                 ├─view-name───────────────────┤  └─┤ correlation-clause ├─┘
                 ├─nickname────────────────────┤
                 └─ONLY──(──┬─table-name─┬──)──┘
                            └─view-name──┘

►─WHERE CURRENT OF──cursor-name──────────────────────────────────────────────────►◄
```

correlation-clause:

```
   ┌─AS─┐
├──┴────┴──correlation-name──┬──────────────────────┬──────────────────────────┤
                             └─(──column-name──)────┘
```

Description:

FROM *table-name*, *view-name*, *nickname*, **or** *(fullselect)*
: Identifies the object of the delete operation. The name must identify a table or view that exists in the catalog, but it must not identify a catalog table, a catalog view, a system-maintained materialized query table, or a read-only view.

  If *table-name* is a typed table, rows of the table or any of its proper subtables may get deleted by the statement.

  If *view-name* is a typed view, rows of the underlying table or underlying tables of the view's proper subviews may get deleted by the statement. If *view-name* is a regular view with an underlying table that is a typed table, rows of the typed table or any of its proper subtables may get deleted by the statement.

If the object of the delete operation is a fullselect, the fullselect must be deletable, as defined in the "Deletable views" Notes item in the description of the CREATE VIEW statement.

Only the columns of the specified table can be referenced in the WHERE clause. For a positioned DELETE, the associated cursor must also have specified the table or view in the FROM clause without using ONLY.

FROM ONLY (*table-name*)
Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

FROM ONLY (*view-name*)
Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be deleted by the statement. For a positioned DELETE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

correlation-clause
Can be used within the *search-condition* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the DELETE statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the DELETE statement.

column-name
Specifies a column of the intermediate result table of the DELETE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

assignment-clause
See the description of *assignment-clause* under the UPDATE statement. The same rules apply. The *include-columns* are the only columns that can be set using the *assignment-clause* (SQLSTATE 42703).

WHERE
Specifies a condition that selects the rows to be deleted. The clause can be omitted, a search condition specified, or a cursor named. If the clause is omitted, all rows of the table or view are deleted.

search-condition
Each *column-name* in the search condition, other than in a subquery must identify a column of the table or view.

The *search-condition* is applied to each row of the table, view, or nickname, and the deleted rows are those for which the result of the *search-condition* is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the *search condition* is applied to a row, and the results used in applying the *search condition*. In actuality, a subquery with no correlated references is executed once, whereas a subquery with a correlated reference may have to be executed once for each row. If a subquery refers to the object table of a DELETE statement or a dependent table with a delete rule of CASCADE or SET NULL, the subquery is completely evaluated before any rows are deleted.

CURRENT OF *cursor-name*
Identifies a cursor that is defined in a DECLARE CURSOR statement of the program. The DECLARE CURSOR statement must precede the DELETE statement.

The table, view, or nickname named must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the DELETE statement is executed, the cursor must be positioned on a row: that row is the one deleted. After the deletion, the cursor is positioned before the next row of its result table. If there is no next row, the cursor is positioned after the last row.

WITH
Specifies the isolation level used when locating the rows to be deleted.

RR
Repeatable Read

RS
Read Stability

CS
Cursor Stability

UR
Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules:

- ***Triggers:*** DELETE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the deleted rows. If a DELETE statement on a view causes an INSTEAD OF trigger to fire, referential integrity will be checked against the updates performed in the trigger, and not against the underlying tables of the view that caused the trigger to fire.
- ***Referential Integrity:*** If the identified table or the base table of the identified view is a parent, the rows selected for delete must not have any dependents in a

relationship with a delete rule of RESTRICT, and the DELETE must not cascade to descendent rows that have dependents in a relationship with a delete rule of RESTRICT.

If the delete operation is not prevented by a RESTRICT delete rule, the selected rows are deleted. Any rows that are dependents of the selected rows are also affected:

– The nullable columns of the foreign keys of any rows that are their dependents in a relationship with a delete rule of SET NULL are set to the null value.
– Any rows that are their dependents in a relationship with a delete rule of CASCADE are also deleted, and the above rules apply, in turn, to those rows.

The delete rule of NO ACTION is checked to enforce that any non-null foreign key refers to an existing parent row after the other referential constraints have been enforced.

Notes:

- If an error occurs during the execution of a multiple row DELETE, no changes are made to the database.
- Unless appropriate locks already exist, one or more exclusive locks are acquired during the execution of a successful DELETE statement. Issuing a COMMIT or ROLLBACK statement will release the locks. Until the locks are released by a commit or rollback operation, the effect of the delete operation can only be perceived by:
  – The application process that performed the deletion
  – Another application process using isolation level UR.

  The locks can prevent other application processes from performing operations on the table.
- If an application process deletes a row on which any of its cursors are positioned, those cursors are positioned before the next row of their result table. Let C be a cursor that is positioned before row R (as a result of an OPEN, a DELETE through C, a DELETE through some other cursor, or a searched DELETE). In the presence of INSERT, UPDATE, and DELETE operations that affect the base table from which R is derived, the next FETCH operation referencing C does not necessarily position C on R. For example, the operation can position C on R′, where R′ is a new row that is now the next row of the result table.
- SQLERRD(3) in the SQLCA shows the number of rows that qualified for the delete operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) in the SQLCA shows the number of rows affected by referential constraints and by triggered statements. It includes rows that were deleted as a result of a CASCADE delete rule and rows in which foreign keys were set to NULL as the result of a SET NULL delete rule. With regards to triggered statements, it includes the number of rows that were inserted, updated, or deleted.
- If an error occurs that prevents deleting all rows matching the search condition and all operations required by existing referential constraints, no changes are made to the table and the error is returned.
- For nicknames, the external server option `iud_app_svpt_enforce` poses an additional limitation. Refer to the Federated documentation for more information.

- For some data sources, the SQLCODE -20190 may be returned on a delete against a nickname because of potential data inconsistency. Refer to the Federated documentation for more information.
- For any deleted row that includes currently linked files through DATALINK columns, the files are unlinked, and will be either restored or deleted, depending on the datalink column definition.

  An error may occur when attempting to delete a DATALINK value if the file server referenced in the value is no longer registered with the database server (SQLSTATE 55022).

  An error may also occur when deleting a row that has a link to a server that is unavailable at the time of deletion (SQLSTATE 57050).

Examples:

Example 1: Delete department (DEPTNO) 'D11' from the DEPARTMENT table.

```
DELETE FROM DEPARTMENT
  WHERE DEPTNO = 'D11'
```

Example 2: Delete all the departments from the DEPARTMENT table (that is, empty the table).

```
DELETE FROM DEPARTMENT
```

Example 3: Delete from the EMPLOYEE table any sales rep or field rep who didn't make a sale in 1995.

```
DELETE FROM EMPLOYEE
  WHERE LASTNAME NOT IN
    (SELECT SALES_PERSON
      FROM SALES
      WHERE YEAR(SALES_DATE)=1995)
      AND JOB IN ('SALESREP','FIELDREP')
```

Example 4: Delete all the duplicate employee rows from the EMPLOYEE table. An employee row is considered to be a duplicate if the last names match. Keep the employee row with the smallest first name in lexical order.

```
DELETE FROM
  (SELECT ROWNUMBER() OVER (PARTITION BY LASTNAME ORDER BY FIRSTNME)
    FROM EMPLOYEE) AS E(RN)
    WHERE RN = 1
```

Related reference:

- "Search conditions" in *SQL Reference, Volume 1*
- "SQLCA (SQL communications area)" on page 1357
- "Subselect" on page 1211
- "CREATE VIEW " on page 1034
- "DECLARE CURSOR statement" in *SQL Reference, Volume 2*
- "UPDATE " on page 1145

Related samples:

- "dbuse.c -- How to use a database"
- "tbmod.c -- How to modify table data"
- "dbuse.sqc -- How to use a database (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbmod.sqc -- How to modify table data (C)"

- "dbuse.sqC -- How to use a database (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "delet.sqb -- How to delete table data (MF COBOL)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"
- "DbUse.java -- How to use a database (JDBC)"
- "TbConstr.java -- How to create, use and drop constraints (JDBC)"
- "TbMod.java -- How to modify table data (JDBC)"
- "DbUse.sqlj -- How to use a database (SQLj)"
- "TbConstr.sqlj -- How to create, use and drop constraints (SQLj)"
- "TbMod.sqlj -- How to modify table data (SQLj)"

DROP

The DROP statement deletes an object. Any objects that are directly or indirectly dependent on that object are either deleted or made inoperative. Whenever an object is deleted, its description is deleted from the catalog, and any packages that reference the object are invalidated.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

When dropping objects that allow two-part names, the privileges held by the authorization ID of the statement must include at least one of the following:
- DROPIN privilege on the schema for the object
- Definer of the object, as recorded in the DEFINER column of the catalog view for the object
- CONTROL privilege on the object (applicable only to indexes, index specifications, nicknames, packages, tables, and views)
- Definer of the user-defined type, as recorded in the DEFINER column of the SYSCAT.DATATYPES catalog view (applicable only when dropping a method that is associated with a user-defined type)
- SYSADM or DBADM authority

When dropping a table or view hierarchy, the privileges held by the authorization ID of the statement must include one of the above privileges for each of the tables or views in the hierarchy.

When dropping a schema, the authorization ID of the statement must hold SYSADM or DBADM authority, or be the schema owner, as recorded in the OWNER column of the SYSCAT.SCHEMATA catalog view.

When dropping a buffer pool, database partition group, or table space, the authorization ID of the statement must hold SYSADM or SYSCTRL authority.

When dropping an event monitor, server definition, data type mapping, function mapping, or wrapper, the authorization ID of the statement must hold SYSADM or DBADM authority.

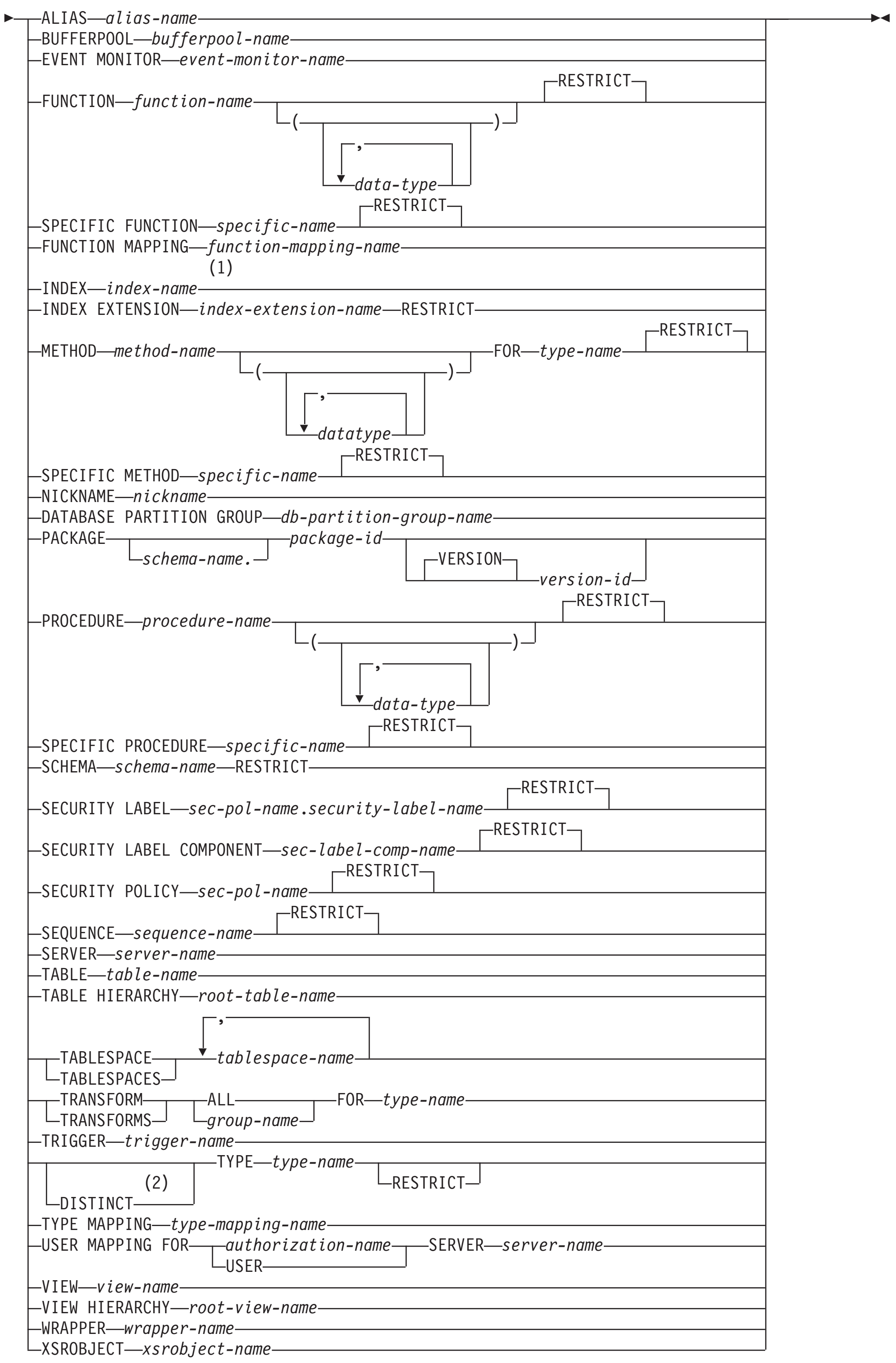
When dropping a user mapping, the authorization ID of the statement must hold SYSADM or DBADM authority, if this authorization ID is different from the federated database authorization name within the mapping. Otherwise, if the authorization ID and the authorization name match, no authorities or privileges are required.

When dropping a transform, the authorization ID of the statement must hold SYSADM or DBADM authority, or must be the definer of *type-name*.

When dropping a security label component, a security label, or a security policy, the privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
►►──DROP──────────────────────────────────────────────────────────────►
```


ALIAS alias-name
BUFFERPOOL bufferpool-name
EVENT MONITOR event-monitor-name
FUNCTION function-name (, data-type) RESTRICT
SPECIFIC FUNCTION specific-name RESTRICT
FUNCTION MAPPING function-mapping-name
INDEX index-name (1)
INDEX EXTENSION index-extension-name RESTRICT
METHOD method-name (, datatype) FOR type-name RESTRICT
SPECIFIC METHOD specific-name RESTRICT
NICKNAME nickname
DATABASE PARTITION GROUP db-partition-group-name
PACKAGE schema-name. package-id VERSION version-id
PROCEDURE procedure-name (, data-type) RESTRICT
SPECIFIC PROCEDURE specific-name RESTRICT
SCHEMA schema-name RESTRICT
SECURITY LABEL sec-pol-name.security-label-name RESTRICT
SECURITY LABEL COMPONENT sec-label-comp-name RESTRICT
SECURITY POLICY sec-pol-name RESTRICT
SEQUENCE sequence-name RESTRICT
SERVER server-name
TABLE table-name
TABLE HIERARCHY root-table-name
TABLESPACE TABLESPACES , tablespace-name
TRANSFORM TRANSFORMS ALL group-name FOR type-name
TRIGGER trigger-name
DISTINCT (2) TYPE type-name RESTRICT
TYPE MAPPING type-mapping-name
USER MAPPING FOR authorization-name USER SERVER server-name
VIEW view-name
VIEW HIERARCHY root-view-name
WRAPPER wrapper-name
XSROBJECT xsrobject-name

Notes:

1 *Index-name* can be the name of either an index or an index specification.

2 DATA can also be used when dropping any user-defined type.

Description:

ALIAS *alias-name*
Identifies the alias that is to be dropped. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704). The specified alias is deleted.

All tables, views, and triggers that reference the alias are made inoperative. (This includes both the table referenced in the ON clause of the CREATE TRIGGER statement, and all tables referenced within the triggered SQL statements.)

BUFFERPOOL *bufferpool-name*
Identifies the buffer pool that is to be dropped. The *bufferpool-name* must identify a buffer pool that is described in the catalog (SQLSTATE 42704). There can be no table spaces assigned to the buffer pool (SQLSTATE 42893). The IBMDEFAULTBP buffer pool cannot be dropped (SQLSTATE 42832). Buffer pool memory is released immediately, to be used by DB2. Disk storage may not be released until the next connection to the database.

EVENT MONITOR *event-monitor-name*
Identifies the event monitor that is to be dropped. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

If the identified event monitor is active, an error is returned (SQLSTATE 55034); otherwise, the event monitor is deleted. Note that if an event monitor has been previously activated using the SET EVENT MONITOR STATE statement, and the database has been deactivated and subsequently reactivated, use the SET EVENT MONITOR STATE statement to deactivate the event monitor before issuing the DROP statement.

If there are event files in the target path of a WRITE TO FILE event monitor that is being dropped, the event files are not deleted. However, if a new event monitor that specifies the same target path is created, the event files are deleted.

When dropping WRITE TO TABLE event monitors, table information is removed from the SYSCAT.EVENTTABLES catalog view, but the tables themselves are not dropped.

FUNCTION
Identifies an instance of a user-defined function (either a complete function or a function template) that is to be dropped. The function instance specified must be a user-defined function described in the catalog. Functions implicitly generated by the CREATE DISTINCT TYPE statement cannot be dropped.

There are several different ways available to identify the function instance:

FUNCTION *function-name*
Identifies the particular function, and is valid only if there is exactly one function instance with the *function-name*. The function thus identified may have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the function in the named or implied schema, an error is returned (SQLSTATE 42725).

FUNCTION *function-name* **(***data-type***,...)**
Provides the function signature, which uniquely identifies the function to be dropped. The function selection algorithm is not used.

function-name
Gives the function name of the function to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Must match the data types that were specified on the CREATE FUNCTION statement in the corresponding position. The number of data types, and the logical concatenation of the data types is used to identify the specific function instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

RESTRICT
The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:
- Another routine is sourced on the function.
- A view uses the function.
- A trigger uses the function.
- A materialized query table uses the function in its definition.

RESTRICT is the default behavior.

If no function with the specified signature exists in named or implied schema, an error is returned (SQLSTATE 42883).

SPECIFIC FUNCTION *specific-name*
Identifies the particular user-defined function that is to be dropped, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

RESTRICT
The RESTRICT keyword enforces the rule that the function is not to be dropped if any of the following dependencies exists:
- Another routine is sourced on the function.

- A view uses the function.
- A trigger uses the function.

RESTRICT is the default behavior.

It is not possible to drop a function that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

Other objects can be dependent upon a function. All such dependencies must be removed before the function can be dropped, with the exception of packages which are marked inoperative. An attempt to drop a function with such dependencies will result in an error (SQLSTATE 42893). See 1070 for a list of these dependencies.

If the function can be dropped, it is dropped.

Any package dependent on the specific function being dropped is marked as inoperative. Such a package is not implicitly rebound. It must either be rebound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping that is to be dropped. The *function-mapping-name* must identify a user-defined function mapping that is described in the catalog (SQLSTATE 42704). The function mapping is deleted from the database.

Default function mappings cannot be dropped, but can be disabled by using the CREATE FUNCTION MAPPING statement. Dropping a user-defined function mapping that was created to override a default function mapping reinstates the default function mapping.

Packages having a dependency on a dropped function mapping are invalidated.

INDEX *index-name*
Identifies the index or index specification that is to be dropped. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704). It cannot be an index that is required by the system for a primary key or unique constraint, for a replicated materialized query table, or for an XML column (SQLSTATE 42917). The specified index or index specification is deleted.

Packages having a dependency on a dropped index or index specification are invalidated.

INDEX EXTENSION *index-extension-name* **RESTRICT**
Identifies the index extension that is to be dropped. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no index can be defined that depends on this index extension definition (SQLSTATE 42893).

METHOD
Identifies a method body that is to be dropped. The method body specified must be a method described in the catalog (SQLSTATE 42704). Method bodies that are implicitly generated by the CREATE TYPE statement cannot be dropped.

DROP METHOD deletes the body of a method, but the method specification (signature) remains as a part of the definition of the subject type. After dropping the body of a method, the method specification can be removed from the subject type definition by ALTER TYPE DROP METHOD.

There are several ways available to identify the method body to be dropped:

METHOD *method-name*
: Identifies the particular method to be dropped, and is valid only if there is exactly one method instance with name *method-name* and subject type *type-name*. Thus, the method identified may have any number of parameters. If no method by this name exists for the type *type-name*, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the method for the named data type, an error is returned (SQLSTATE 42725).

METHOD *method-name* **(***data-type***,...)**
: Provides the method signature, which uniquely identifies the method to be dropped. The method selection algorithm is not used.

  method-name
  : The method name of the method to be dropped for the specified type. The name must be an unqualified identifier.

  (*data-type***, ...)**
  : Must match the data types that were specified in the corresponding positions of the method-specification of the CREATE TYPE or ALTER TYPE statement. The number of data types and the logical concatenation of the data types are used to identify the specific method instance which is to be dropped.

    If the data-type is unqualified, the type name is resolved by searching the schemas on the SQL path.

    It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

    FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

    However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

    A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

    If no method with the specified signature exists for the named data type, an error is returned (SQLSTATE 42883).

FOR *type-name*
: Names the type for which the specified method is to be dropped. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names.

RESTRICT
: The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

  - Another routine is sourced on the method.

- A view uses the method.
- A trigger uses the method.

RESTRICT is the default behavior.

SPECIFIC METHOD *specific-name*

Identifies the particular method that is to be dropped, using a name either specified or defaulted to at CREATE TYPE or ALTER TYPE time. If the specific name is unqualified, the CURRENT SCHEMA special register is used as a qualifier for an unqualified specific name in dynamic SQL. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for an unqualified specific name. The specific-name must identify a method; otherwise, an error is returned (SQLSTATE 42704).

RESTRICT

The RESTRICT keyword enforces the rule that the method is not to be dropped if any of the following dependencies exists:

- Another routine is sourced on the method.
- A view uses the method.
- A trigger uses the function.

RESTRICT is the default method.

Other objects can be dependent upon a method. All such dependencies must be removed before the method can be dropped, with the exception of packages which will be marked inoperative if the drop is successful. An attempt to drop a method with such dependencies will result in an error (SQLSTATE 42893).

If the method can be dropped, it will be dropped.

Any package dependent on the specific method being dropped is marked as inoperative. Such a package is not implicitly re-bound. Either it must be re-bound by use of the BIND or REBIND command, or it must be re-prepared by use of the PREP command.

If the specific method being dropped overrides another method, all packages dependent on the overridden method — and on methods that override this method in supertypes of the specific method being dropped — are invalidated.

NICKNAME *nickname*

Identifies the nickname that is to be dropped. The nickname must be listed in the catalog (SQLSTATE 42704). The nickname is deleted from the database.

All information about the columns and indexes associated with the nickname is deleted from the catalog. Any materialized query tables that are dependent on the nickname are dropped. Any index specifications that are dependent on the nickname are dropped. Any views that are dependent on the nickname are marked inoperative. Any packages that are dependent on the dropped index specifications or inoperative views are invalidated. The data source table that the nickname references is not affected.

If an SQL function or method is dependent on a nickname, that nickname cannot be dropped (SQLSTATE 42893).

DATABASE PARTITION GROUP *db-partition-group-name*

Identifies the database partition group that is to be dropped. The *db-partition-group-name* parameter must identify a database partition group that is described in the catalog (SQLSTATE 42704). This is a one-part name.

Dropping a database partition group drops all table spaces defined in the database partition group. All existing database objects with dependencies on the tables in the table spaces (such as packages, referential constraints, and so on) are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

System-defined database partition groups cannot be dropped (SQLSTATE 42832).

If a DROP DATABASE PARTITION GROUP statement is issued against a database partition group that is currently undergoing a data redistribution, the drop database partition group operation fails, and an error is returned (SQLSTATE 55038). However, a partially redistributed database partition group can be dropped. A database partition group can become partially redistributed if a REDISTRIBUTE DATABASE PARTITION GROUP command does not execute to completion. This can happen if it is interrupted by either an error or a FORCE APPLICATION ALL command. (For a partially redistributed database partition group, the REBALANCE_PMAP_ID in the SYSCAT.DBPARTITIONGROUPS catalog is not −1.)

PACKAGE *schema-name.package-id*
Identifies the package that is to be dropped. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704). The specified package is deleted. If the package being dropped is the only package identified by *schema-name.package-id* (that is, there are no other versions), all privileges on the package are also deleted.

VERSION *version-id*
Identifies which package version is to be dropped. If a value is not specified, the version defaults to the empty string. If multiple packages with the same package name but different versions exist, only one package version can be dropped in one invocation of the DROP statement. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

PROCEDURE
Identifies an instance of a procedure that is to be dropped. The procedure instance specified must be a procedure described in the catalog.

There are several different ways available to identify the procedure instance:

PROCEDURE *procedure-name*
Identifies the particular procedure to be dropped, and is valid only if there is exactly one procedure instance with the *procedure-name* in the schema. The procedure thus identified may have any number of parameters defined for it. If no procedure by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified

object names. If there is more than one specific instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is returned.

PROCEDURE *procedure-name* **(***data-type***,...)**
Provides the procedure signature, which uniquely identifies the procedure to be dropped. The procedure selection algorithm is not used. For federated procedures, the signature information is not specified on the CREATE PROCEDURE statement, but the information is available in the system catalog.

procedure-name
Gives the procedure name of the procedure to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Must match the data types that were specified on the CREATE PROCEDURE statement in the corresponding position, except for federated procedures, where the data type must match what is stored in the local catalog for the corresponding parameter. The number of data types, and the logical concatenation of the data types is used to identify the specific procedure instance which is to be dropped.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision or scale for the parameterized data types. Instead, an empty set of parentheses may be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601) since the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement or, for federated procedures, it must exactly match what is stored in the local catalog for the corresponding parameter.

A type of FLOAT(n) does not need to match the defined value for n since 0<n<25 means REAL and 24<n<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

RESTRICT
The RESTRICT keyword prevents the procedure from being dropped if a trigger definition, an SQL function, or an SQL method contains a CALL statement with the name of the procedure. RESTRICT is the default behavior.

If no procedure with the specified signature exists in named or implied schema, an error (SQLSTATE 42883) is returned.

SPECIFIC PROCEDURE *specific-name*
Identifies the particular procedure that is to be dropped, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object

names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

RESTRICT
The RESTRICT keyword prevents the procedure from being dropped if a trigger definition, an SQL function, or an SQL method contains a CALL statement with the name of the procedure. RESTRICT is the default behavior.

It is not possible to drop a procedure that is in the SYSIBM, SYSFUN, or the SYSPROC schema (SQLSTATE 42832).

SCHEMA *schema-name* **RESTRICT**
Identifies the particular schema to be dropped. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704). The RESTRICT keyword enforces the rule that no objects can be defined in the specified schema for the schema to be deleted from the database (SQLSTATE 42893).

SECURITY LABEL *sec-pol-name***.***security-label-name*
Identifies the security label to be dropped. The *security-label-name* must identify a security label that is described in the catalog (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the security label from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more users currently hold the security label for read access
- One or more users currently hold the security label for write access
- The security label is currently being used to protect one or more columns
- The security label is currently being used to protect one or more rows

SECURITY LABEL COMPONENT *sec-label-comp-name*
Identifies the security label component to be dropped. The *sec-label-comp-name* must identify a security label component that is described in the catalog (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the security label component from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more security policies that include the security label component are currently defined

SECURITY POLICY *sec-pol-name*
Identifies the security policy to be dropped. The *sec-pol-name* must identify a security policy that is described in the catalog (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the security policy from being dropped if any of the following dependencies exist (SQLSTATE 42893):

- One or more tables are associated with this security policy
- One or more users hold an exemption to one of the rules in this security policy

SEQUENCE *sequence-name*
Identifies the particular sequence that is to be dropped. The *sequence-name*, along with the implicit or explicit schema name, must identify an existing

sequence at the current server. If no sequence by this name exists in the explicitly or implicitly specified schema, an error is returned (SQLSTATE 42704).

RESTRICT
This option, which is the default, prevents the sequence from being dropped if any of the following dependencies exist:

- A trigger exists such that a NEXT VALUE or PREVIOUS VALUE expression in the trigger specifies the sequence (SQLSTATE 42893).
- An SQL function or an SQL method exists such that a NEXT VALUE expression in the routine body specifies the sequence (SQLSTATE 42893).

SERVER *server-name*
Identifies the data source whose definition is to be dropped from the catalog. The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The definition of the data source is deleted.

All nicknames for tables and views residing at the data source are dropped. Any index specifications dependent on these nicknames are dropped. Any user-defined function mappings, user-defined type mappings, and user mappings that are dependent on the dropped server definition are also dropped. All packages dependent on the dropped server definition, function mappings, nicknames, and index specifications are invalidated. All federated procedures that are dependent on the server definition are also dropped.

TABLE *table-name*
Identifies the base table, declared temporary table, or nickname that is to be dropped. The *table-name* must identify a table that is described in the catalog or, if it is a declared temporary table, the *table-name* must be qualified by the schema name SESSION and exist in the application (SQLSTATE 42704). The subtables of a typed table are dependent on their supertables. All subtables must be dropped before a supertable can be dropped (SQLSTATE 42893). The specified table is deleted from the database.

All indexes, primary keys, foreign keys, check constraints, materialized query tables, and staging tables referencing the table are dropped. All views and triggers that reference the table are made inoperative. (This includes both the table referenced in the ON clause of the CREATE TRIGGER statement, and all tables referenced within the triggered SQL statements.) All packages depending on any object dropped or marked inoperative will be invalidated. This includes packages dependent on any supertables above the subtable in the hierarchy. Any reference columns for which the dropped table is defined as the scope of the reference become unscoped.

Packages are not dependent on declared temporary tables, and therefore are not invalidated when such a table is dropped.

All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously so the files may not be immediately available for other operations.

In a federated system, a remote table that was created using transparent DDL can be dropped. Dropping a remote table also drops the nickname associated with that table, and invalidates any packages that are dependent on that nickname.

When a subtable is dropped from a table hierarchy, the columns associated with the subtable are no longer accessible although they continue to be considered with respect to limits on the number of columns and size of the row. Dropping a subtable has the effect of deleting all the rows of the subtable from the supertables. This may result in activation of triggers or referential integrity constraints defined on the supertables.

When a declared temporary table is dropped, and its creation preceded the active unit of work or savepoint, then the table will be functionally dropped and the application will not be able to access the table. However, the table will still reserve some space in its table space and will prevent that USER TEMPORARY table space from being dropped or the database partition group of the USER TEMPORARY table space from being redistributed until the unit of work is committed or savepoint is ended. Dropping a declared temporary table causes the data in the table to be destroyed, regardless of whether DROP is committed or rolled back.

A table cannot be dropped if it has the RESTRICT ON DROP attribute.

A newly detached table is initially inaccessible. This prevents the table from being read, modified, or dropped until the SET INTEGRITY statement can be run to incrementally refresh MQTs or to complete any processing for foreign key constraints. After the SET INTEGRITY statement executes against all dependent tables, the table is fully accessible, its detached attribute is reset, and it can be dropped.

TABLE HIERARCHY *root-table-name*
Identifies the typed table hierarchy that is to be dropped. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR). The typed table identified by *root-table-name* and all of its subtables are deleted from the database.

All indexes, materialized query tables, staging tables, primary keys, foreign keys, and check constraints referencing the dropped tables are dropped. All views and triggers that reference the dropped tables are made inoperative. All packages depending on any object dropped or marked inoperative will be invalidated. Any reference columns for which one of the dropped tables is defined as the scope of the reference become unscoped.

All files that are linked through any DATALINK columns are unlinked. The unlink operation is performed asynchronously so the files may not be immediately available for other operations.

Unlike dropping a single subtable, dropping the table hierarchy does not result in the activation of delete triggers of any tables in the hierarchy nor does it log the deleted rows.

TABLESPACE or **TABLESPACES** *tablespace-name*
Identifies the table spaces that are to be dropped; *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704). This is a one-part name.

The table spaces will not be dropped (SQLSTATE 55024) if there is any table that stores at least one of its parts in a table space being dropped, and has one or more of its parts in another table space that is not being dropped (these tables would need to be dropped first), or if any table that resides in the table space has the RESTRICT ON DROP attribute.

Objects whose names are prefixed with 'SYS' are system-defined objects and, with the exception of the SYSTOOLSPACE and SYSTOOLSTMPSPACE table spaces, cannot be dropped (SQLSTATE 42832).

A SYSTEM TEMPORARY table space cannot be dropped (SQLSTATE 55026) if it is the only temporary table space that exists in the database. A USER TEMPORARY table space cannot be dropped if there is a declared temporary table created in it (SQLSTATE 55039). Even if a declared temporary table has been dropped, the USER TEMPORARY table space will still be considered to be in use until the unit of work containing the DROP TABLE statement has been committed.

Dropping a table space drops all objects that are defined in the table space. All existing database objects with dependencies on the table space, such as packages, referential constraints, and so on, are dropped or invalidated (as appropriate), and dependent views and triggers are made inoperative.

Containers that were created by a user are not deleted. Any directories in the path of the container name that were created by the database manager during CREATE TABLESPACE execution are deleted. All containers that are below the database directory are deleted. When the DROP TABLESPACE statement is committed, the DMS file containers or SMS containers for the specified table space are deleted, if possible. If the containers cannot be deleted (because they are being kept open by another agent, for example), the files are truncated to zero length. After all connections are terminated, or the DEACTIVATE DATABASE command is issued, these zero-length files are deleted.

TRANSFORM ALL FOR *type-name*

Indicates that all transforms groups defined for the user-defined data type *type-name* are to be dropped. The transform functions referenced in these groups are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704).

If there are not transforms defined for *type-name*, an error is returned (SQLSTATE 42740).

DROP TRANSFORM is the inverse of CREATE TRANSFORM. It causes the transform functions associated with certain groups, for a given datatype, to become undefined. The functions formerly associated with these groups still exist and can still be called explicitly, but they no longer have the transform property, and are no longer invoked implicitly for exchanging values with the host language environment.

The transform group is not dropped if there is a user-defined function (or method) written in a language other than SQL that has a dependency on one of the group's transform functions defined for the user-defined type *type-name* (SQLSTATE 42893). Such a function has a dependency on the transform function associated with the referenced transform group defined for type *type-name*. Packages that depend on a transform function associated with the named transform group are marked inoperative.

TRANSFORMS *group-name* **FOR** *type-name*

Indicates that the specified transform group for the user-defined data type *type-name* is to be dropped. The transform functions referenced in this group are not dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *type-name* must identify a user-defined type described in the catalog (SQLSTATE 42704), and the *group-name* must identify an existing transform group for *type-name*.

TRIGGER *trigger-name*
Identifies the trigger that is to be dropped. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704). The specified trigger is deleted.

Dropping triggers causes certain packages to be marked invalid.

If *trigger-name* specifies an INSTEAD OF trigger on a view, another trigger may depend on that trigger through an update against the view.

TYPE *type-name*
Identifies the user-defined type to be dropped. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. For a structured type, the associated reference type is also dropped. The *type-name* must identify a user-defined type described in the catalog. If DISTINCT is specified, then the *type-name* must identify a distinct type described in the catalog.

RESTRICT
The type is not dropped (SQLSTATE 42893) if any of the following is true:

- The type is used as the type of a column of a table or view.
- The type has a subtype.
- The type is a structured type used as the data type of a typed table or a typed view.
- The type is an attribute of another structured type.
- There exists a column of a table whose type might contain an instance of *type-name*. This can occur if *type-name* is the type of the column or is used elsewhere in the column's associated type hierarchy. More formally, for any type T, T cannot be dropped if there exists a column of a table whose type directly or indirectly uses *type-name*.
- The type is the target type of a reference-type column of a table or view, or a reference-type attribute of another structured type.
- The type, or a reference to the type, is a parameter type or a return value type of a function or method.
- The type, or a reference to the type, is used in the body of an SQL function or method, but it is not a parameter type or a return value type.
- The type is used in a check constraint, trigger, view definition, or index extension.

If RESTRICT is not specified, the behavior is the same as RESTRICT, except for functions and methods that use the type.

Functions that use the type: If the user-defined type can be dropped, then for every function, F (with specific name SF), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following DROP FUNCTION statement is effectively executed:

```
DROP SPECIFIC FUNCTION SF
```

It is possible that this statement also would cascade to drop dependent functions. If all of these functions are also in the list to be dropped because of a dependency on the user-defined type, the drop of the user-defined type will succeed (otherwise it fails with SQLSTATE 42893).

Methods that use the type: If the user-defined type can be dropped, then for every method, M of type T1 (with specific name SM), that has parameters or a return value of the type being dropped or a reference to the type being dropped, the following statements are effectively executed:

```
DROP SPECIFIC METHOD SM
ALTER TYPE T1 DROP SPECIFIC METHOD SM
```

The existence of objects that are dependent on these methods may cause the DROP TYPE operation to fail.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

TYPE MAPPING *type-mapping-name*
Identifies the user-defined data type mapping to be dropped. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704). The data type mapping is deleted from the database.

No additional objects are dropped.

USER MAPPING FOR *authorization-name* **| USER SERVER** *server-name*
Identifies the user mapping to be dropped. This mapping associates an authorization name that is used to access the federated database with an authorization name that is used to access a data source. The first of these two authorization names is either identified by the *authorization-name* or referenced by the special register USER. The *server-name* identifies the data source that the second authorization name is used to access.

The *authorization-name* must be listed in the catalog (SQLSTATE 42704). The *server-name* must identify a data source that is described in the catalog (SQLSTATE 42704). The user mapping is deleted.

No additional objects are dropped.

VIEW *view-name*
Identifies the view that is to be dropped. The *view-name* must identify a view that is described in the catalog (SQLSTATE 42704). The subviews of a typed view are dependent on their superviews. All subviews must be dropped before a superview can be dropped (SQLSTATE 42893).

The specified view is deleted. The definition of any view or trigger that is directly or indirectly dependent on that view is marked inoperative. Any materialized query table or staging table that is dependent on any view that is marked inoperative is dropped. Any packages dependent on a view that is dropped or marked inoperative will be invalidated. This includes packages dependent on any superviews above the subview in the hierarchy. Any reference columns for which the dropped view is defined as the scope of the reference become unscoped.

VIEW HIERARCHY *root-view-name*
Identifies the typed view hierarchy that is to be dropped. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR). The typed view identified by *root-view-name* and all of its subviews are deleted from the database.

The definition of any view or trigger that is directly or indirectly dependent on any of the dropped views is marked inoperative. Any packages dependent on any view or trigger that is dropped or marked inoperative will be invalidated.

Any reference columns for which a dropped view or view marked inoperative is defined as the scope of the reference become unscoped.

WRAPPER *wrapper-name*
Identifies the wrapper to be dropped. The *wrapper-name* must identify a wrapper that is described in the catalog (SQLSTATE 42704). The wrapper is deleted.

All server definitions, user-defined function mappings, and user-defined data type mappings that are dependent on the wrapper are dropped. All user-defined function mappings, nicknames, user-defined data type mappings, and user mappings that are dependent on the dropped server definitions are also dropped. Any index specifications dependent on the dropped nicknames are dropped, and any views dependent on these nicknames are marked inoperative. All packages dependent on the dropped objects and inoperative views are invalidated. All federated procedures that are dependent on the dropped server definitions are also dropped.

XSROBJECT *xsrobject-name*
Identifies the XSR object to be dropped. The *xsrobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704).

All views referencing the XSR object are marked inoperative. Packages having a dependency on a dropped XSR object are invalidated.

Rules:

Dependencies: Table 145 on page 1071 shows the dependencies that objects have on each other. Not all dependencies are explicitly recorded in the catalog. For example, there is no record of the constraints on which a package has dependencies. Four different types of dependencies are shown:

R Restrict semantics. The underlying object cannot be dropped as long as the object that depends on it exists.

C Cascade semantics. Dropping the underlying object causes the object that depends on it (the depending object) to be dropped as well. However, if the depending object cannot be dropped because it has a Restrict dependency on some other object, the drop of the underlying object will fail.

X Inoperative semantics. Dropping the underlying object causes the object that depends on it to become inoperative. It remains inoperative until a user takes some explicit action.

A Automatic Invalidation/Revalidation semantics. Dropping the underlying object causes the object that depends on it to become invalid. The database manager attempts to revalidate the invalid object.

A package used by a function or a method, or by a procedure that is called directly or indirectly from a function or method, will only be automatically revalidated if the routine is defined as MODIFIES SQL DATA. If the routine is not MODIFIES SQL DATA, an error is returned (SQLSTATE 56098).

Some DROP statement parameters and objects are not shown in Table 145 on page 1071 because they would result in blank rows or columns:

- EVENT MONITOR, PACKAGE, PROCEDURE, SCHEMA, TYPE MAPPING, and USER MAPPING DROP statements do not have object dependencies.

- Alias, buffer pool, distribution key, privilege, and procedure object types do not have DROP statement dependencies.
- A DROP SERVER, DROP FUNCTION MAPPING, or DROP TYPE MAPPING statement in a given unit of work (UOW) cannot be processed under either of the following conditions:
  - The statement references a single data source, and the UOW already includes a SELECT statement that references a nickname for a table or view within this data source (SQLSTATE 55006).
  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes a SELECT statement that references a nickname for a table or view within one of these data sources (SQLSTATE 55006).

Table 145. Dependencies

	Object Type																	
Statement	**CONSTRAINT**	**FUNCTION**	**FUNCTION MAPPING**	**INDEX**	**INDEX EXTENSION**	**METHOD**	**NICKNAME**	**DB PARTITION GROUP**	**PACKAGE**[31]	**SERVER**	**TABLE**	**TABLE SPACE**	**TRIGGER**	**TYPE**	**TYPE MAPPING**	**USER MAPPING**	**VIEW**	**XSROBJECT**
ALTER FUNCTION	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
ALTER METHOD	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
ALTER NICKNAME, altering the local name or the local type	R[33]	R	-	-	-	R	-	-	A	-	R	-	-	-	-	-	R	-
ALTER NICKNAME, altering a column option or a nickname option	-	-	-	-	-	-	-	-	A	-	R	-	-	-	-	-	-	-
ALTER NICKNAME, adding, altering, or dropping a constraint	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
ALTER PROCEDURE	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
ALTER SERVER	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
ALTER TABLE ALTER COLUMN	-	A	-	-	-	-	-	-	A	-	-	-	A	-	-	-	A	X[34]
ALTER TABLE DROP COLUMN	C	C	-	C	-	-	-	-	-	-	-	-	C	-	-	-	C	X[34]
ALTER TABLE DROP CONSTRAINT	C	-	-	-	-	-	-	-	A[1]	-	-	-	-	-	-	-	-	-
ALTER TABLE DROP PARTITIONING KEY	-	-	-	-	-	-	-	R[20]	A[1]	-	-	-	-	-	-	-	-	-
ALTER TYPE ADD ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]	-

Table 145. Dependencies (continued)

	Object Type																	
Statement	**CONSTRAINT**	**FUNCTION**	**FUNCTION MAPPING**	**INDEX**	**INDEX EXTENSION**	**METHOD**	**NICKNAME**	**DB PARTITION GROUP**	**PACKAGE[31]**	**SERVER**	**TABLE**	**TABLE SPACE**	**TRIGGER**	**TYPE**	**TYPE MAPPING**	**USER MAPPING**	**VIEW**	**XSROBJECT**
ALTER TYPE ALTER METHOD	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP ATTRIBUTE	-	-	-	-	R	-	-	-	A[23]	-	R[24]	-	-	-	-	-	R[14]	-
ALTER TYPE ADD METHOD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ALTER TYPE DROP METHOD	-	-	-	-	-	R[27]	-	-	-	-	-	-	-	-	-	-	-	-
CREATE METHOD	-	-	-	-	-	-	-	-	A[28]	-	-	-	-	-	-	-	-	-
CREATE TYPE	-	-	-	-	-	-	-	-	A[29]	-	-	-	-	-	-	-	-	-
DROP ALIAS	-	R	-	-	-	-	-	-	A[3]	-	R[3]	-	X[3]	-	-	-	X[3]	-
DROP BUFFERPOOL	-	-	-	-	-	-	-	-	-	-	-	R	-	-	-	-	-	-
DROP DATABASE PARTITION GROUP	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-	-	-	-
DROP FUNCTION	R	R[7]	R	-	R	R[7]	-	-	X	-	R	-	R	-	-	-	R	-
DROP FUNCTION MAPPING	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	-	-
DROP INDEX	R	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	R[17]	-
DROP INDEX EXTENSION	-	R	-	R	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DROP METHOD	R	R[7]	R	-	R	R	-	-	X/A[30]	-	R	-	R	-	-	-	R	-
DROP NICKNAME	-	R	-	C	-	R	-	-	A	-	C[11]	-	-	-	-	-	X[16]	-
DROP PROCEDURE	-	R[7]	-	-	-	R[7]	-	-	A	-	-	-	R	-	-	-	-	-
DROP SEQUENCE	-	R	-	-	-	R	-	-	A	-	-	-	R	-	-	-	-	-
DROP SERVER	-	C[21]	C[19]	-	-	-	C	-	A	-	-	-	-	-	C[19]	C	-	-
DROP TABLE[32]	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]	X[34]
DROP TABLE HIERARCHY	C	R	-	C	-	-	-	-	A[9]	-	RC[11]	-	X[16]	-	-	-	X[16]	-
DROP TABLESPACE	-	-	-	C[6]	-	-	-	-	-	-	CR[6]	-	-	-	-	-	-	-
DROP TRANSFORM	-	R	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
DROP TRIGGER	-	-	-	-	-	-	-	-	A[1]	-	-	-	X[26]	-	-	-	-	-
DROP TYPE	R[13]	R[5]	-	-	R	-	-	-	A[12]	-	R[18]	-	R[13]	R[4]	-	-	R[14]	-
DROP VIEW	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[15]	-
DROP VIEW HIERARCHY	-	R	-	-	-	-	-	-	A[2]	-	-	-	X[16]	-	-	-	X[16]	-

Table 145. Dependencies (continued)

	Object Type																	
Statement	CONSTRAINT	FUNCTION	FUNCTION MAPPING	INDEX	INDEX EXTENSION	METHOD	NICKNAME	DB PARTITION GROUP	PACKAGE[31]	SERVER	TABLE	TABLE SPACE	TRIGGER	TYPE	TYPE MAPPING	USER MAPPING	VIEW	XSROBJECT
DROP WRAPPER	-	-	C	-	-	-	-	-	-	C	-	-	-	-	C	-	-	-
DROP XSROBJECT	-	-	-	-	-	-	-	-	A	-	-	-	-	-	-	-	X	-
REVOKE a privilege[10]	-	CR[25]	-	-	-	CR[25]	-	-	A[1]	-	CX[8]	-	X	-	-	-	X[8]	-

1 This dependency is implicit in depending on a table with these constraints, triggers, or a distribution key.

2 If a package has an INSERT, UPDATE, or DELETE statement acting upon a view, then the package has an insert, update or delete usage on the underlying base table of the view. In the case of UPDATE, the package has an update usage on each column of the underlying base table that is modified by the UPDATE.

If a package has a statement acting on a typed view, creating or dropping any view in the same view hierarchy will invalidate the package.

3 If a package, materialized query table, staging table, view, or trigger uses an alias, it becomes dependent both on the alias and the object that the alias references. If the alias is in a chain, a dependency is created on each alias in the chain.

Aliases themselves are not dependent on anything. It is possible for an alias to be defined on an object that does not exist.

4 A user-defined type T can depend on another user-defined type B, if T:

- names B as the data type of an attribute
- has an attribute of REF(B)
- has B as a supertype.

5 Dropping a data type cascades to drop the functions and methods that use that data type as a parameter or a result type, and methods defined on the data type. Dropping of these functions and methods will not be prevented by the fact that they depend on each other. However, for functions or methods using the datatype within their bodies, restrict semantics apply.

6 Dropping a table space or a list of table spaces causes all the tables that are completely contained within the given table space or list to be dropped. However, if a table spans table spaces (indexes, long columns, or data

partitions in different table spaces) and those table spaces are not in the list being dropped, the table spaces cannot be dropped as long as the table exists.

7 A function can depend on another specific function if the depending function names the base function in a SOURCE clause. A function or method can also depend on another specific function or method if the depending routine is written in SQL and uses the base routine in its body. An external method, or an external function with a structured type parameter or returns type will also depend on one or more transform functions.

8 Only loss of SELECT privilege will cause a materialized query table to be dropped or a view to become inoperative. If the view that is made inoperative is included in a typed view hierarchy, all of its subviews also become inoperative.

9 If a package has an INSERT, UPDATE, or DELETE statement acting on table T, then the package has an insert, update or delete usage on T. In the case of UPDATE, the package has an update usage on each column of T that is modified by the UPDATE.

If a package has a statement acting on a typed table, creating or dropping any table in the same table hierarchy will invalidate the package.

10 Dependencies do not exist at the column level because privileges on columns cannot be revoked individually.

If a package, trigger or view includes the use of OUTER(*Z*) in the FROM clause, there is a dependency on the SELECT privilege on every subtable or subview of *Z*. Similarly, if a package, trigger, or view includes the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, there is a dependency on the SELECT privilege on every subtable or subview of *Z*.

11 A materialized query table is dependent on the underlying tables or nicknames specified in the fullselect of the table definition.

Cascade semantics apply to dependent materialized query tables.

A subtable is dependent on its supertables up to the root table. A supertable cannot be dropped until all of its subtables are dropped.

12 A package can depend on structured types as a result of using the TYPE predicate or the subtype-treatment expression (TREAT *expression* AS *data-type*). The package has a dependency on the subtypes of each structured type specified in the right side of the TYPE predicate, or the right side of the TREAT expression. Dropping or creating a structured type that alters the subtypes on which the package is dependent causes invalidation.

All packages that are dependent on methods defined in supertypes of the type being dropped, and that are eligible for overriding, are invalidated.

13 A check constraint or trigger is dependent on a type if the type is used anywhere in the constraint or trigger. There is no dependency on the subtypes of a structured type used in a TYPE predicate within a check constraint or trigger.

14 A view is dependent on a type if the type is used anywhere in the view

definition (this includes the type of typed view). There is no dependency on the subtypes of a structured type used in a TYPE predicate within a view definition.

15 A subview is dependent on its superview up to the root view. A superview cannot be dropped until all its subviews are dropped. Refer to [16] for additional view dependencies.

16 A trigger or view is also dependent on the target table or target view of a dereference operation or DEREF function. A trigger or view with a FROM clause that includes OUTER(Z) is dependent on all the subtables or subviews of Z that existed at the time the trigger or view was created.

17 A typed view can depend on the existence of a unique index to ensure the uniqueness of the object identifier column.

18 A table may depend on a user defined data type (distinct or structured) because the type is:

- used as the type of a column
- used as the type of the table
- used as an attribute of the type of the table
- used as the target type of a reference type that is the type of a column of the table or an attribute of the type of the table
- directly or indirectly used by a type that is the column of the table.

19 Dropping a server cascades to drop the function mappings and type mappings created for that named server.

20 If the distribution key is defined on a table in a multiple partition database partition group, the distribution key is required.

21 If a dependent OLE DB table function has ″R″ dependent objects (see DROP FUNCTION), then the server cannot be dropped.

22 An SQL function or method can depend on the objects referenced by its body.

23 When an attribute A of type TA of *type-name* T is dropped, the following DROP statements are effectively executed:

```
Mutator method: DROP METHOD A (TA) FOR T
Observer method: DROP METHOD A () FOR T
ALTER TYPE T
  DROP METHOD A(TA)
  DROP METHOD A()
```

24 A table may depend on an attribute of a user-defined structured data type in the following cases:

1. The table is a typed table that is based on *type-name* or any of its subtypes.
2. The table has an existing column of a type that directly or indirectly refers to *type-name*.

25 A REVOKE of SELECT privilege on a table or view that is used in the body of an SQL function or method body causes an attempt to drop the function or method body, if the function or method body defined no longer has the SELECT privilege. If such a function or method body is used in a view, trigger, function, or method body, it cannot be dropped, and the REVOKE is restricted as a result. Otherwise, the REVOKE cascades and drops such functions.

26 A trigger depends on an INSTEAD OF trigger when it modifies the view on which the INSTEAD OF trigger is defined, and the INSTEAD OF trigger fires.

27 A method declaration of an original method that is overridden by other methods cannot be dropped.(SQLSTATE -2).

28 If the method of the method body being created is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the method being created, are invalidated.

29 When a new subtype of an existing type is created, all packages dependent on methods that are defined in supertypes of the type being created, and that are eligible for overriding (for example, no mutators or observers), are invalidated.

30 If the specific method of the method body being dropped is declared to override another method, all packages dependent on the overridden method, and on methods that override this method in supertypes of the specific method being dropped, are invalidated.

31 Cached dynamic SQL has the same semantics as packages.

32 When a remote base table is dropped using the DROP TABLE statement, both the nickname and the remote base table are dropped.

33 A primary key or unique keys that are not referenced by a foreign key do not restrict the altering of a nickname local name or local type.

34 An XSROBJECT can become inoperative for decomposition as a result of changes to a table that is associated with the XML schema for decomposition. Changes that could impact decomposition are: dropping the table or dropping a column of the table, or changing a column of the table. The decomposition status of the XML schema can be reset by issuing an ALTER XSROBJECT statement to enable or disable decomposition for the XML schema.

Notes:

- It is valid to drop a user-defined function while it is in use. Also, a cursor can be open over a statement which contains a reference to a user-defined function, and while this cursor is open the function can be dropped without causing the cursor fetches to fail.
- If a package which depends on a user-defined function is executing, it is not possible for another authorization ID to drop the function until the package completes its current unit of work. At that point, the function is dropped and the package becomes inoperative. The next request for this package results in an error indicating that the package must be explicitly rebound.
- The removal of a function body (this is very different from dropping the function) can occur while an application which needs the function body is executing. This may or may not cause the statement to fail, depending on whether the function body still needs to be loaded into storage by the database manager on behalf of the statement.
- For any dropped table that includes currently linked files through DATALINK columns, the files are unlinked, and will be either restored or deleted, depending on the datalink column definition.
- If a table containing a DATALINK column is dropped while any DB2 Data Links Managers configured to the database are unavailable, either through DROP TABLE or DROP TABLESPACE, then the operation will fail (SQLSTATE 57050).

- In addition to the dependencies recorded for any explicitly specified UDF, the following dependencies are recorded when transforms are implicitly required:
  1. When the structured type parameter or result of a function or method requires a transform, a dependency is recorded for the function or method on the required TO SQL or FROM SQL transform function.
  2. When an SQL statement included in a package requires a transform function, a dependency is recorded for the package on the designated TO SQL or FROM SQL transform function.

  Since the above describes the only circumstances under which dependencies are recorded due to implicit invocation of transforms, no objects other than functions, methods, or packages can have a dependency on implicitly invoked transform functions. On the other hand, explicit calls to transform functions (in views and triggers, for example) do result in the usual dependencies of these other types of objects on transform functions. As a result, a DROP TRANSFORM statement may also fail due to these ″explicit″ type dependencies of objects on the transform(s) being dropped (SQLSTATE 42893).
- Since the dependency catalogs do not distinguish between depending on a function as a transform versus depending on a function by explicit function call, it is suggested that explicit calls to transform functions are not written. In such an instance, the transform property on the function cannot be dropped, or packages will be marked inoperative, simply because they contain explicit invocations in an SQL expression.
- System created sequences for IDENTITY columns cannot be dropped using the DROP SEQUENCE statement.
- When a sequence is dropped, all privileges on the sequence are also dropped and any packages that refer to the sequence are invalidated.
- For relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under either of the following conditions (SQLSTATE 55007):
  – A nickname referenced in this statement has a cursor open on it in the same UOW
  – Either an INSERT, DELETE, or UPDATE statement is already issued in the same UOW against the nickname that is referenced in this statement
- For non-relational nicknames, the DROP NICKNAME statement within a given unit of work (UOW) cannot be processed under any of the following conditions (SQLSTATE 55007):
  – A nickname referenced in this statement has a cursor open on it in the same UOW
  – A nickname referenced in this statement is already referenced by a SELECT statement in the same UOW
  – Either an INSERT, DELETE, or UPDATE statement has already been issued in the same UOW against the nickname that is referenced in this statement
- A DROP SERVER statement (SQLSTATE 55006), or a DROP FUNCTION MAPPING or DROP TYPE MAPPING statement (SQLSTATE 55007) within a given unit of work (UOW) cannot be processed under either of the following conditions:
  – The statement references a single data source, and the UOW already includes one of the following:
    - A SELECT statement that references a nickname for a table or view within this data source
    - An open cursor on a nickname for a table or view within this data source

      - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within this data source
  - The statement references a category of data sources (for example, all data sources of a specific type and version), and the UOW already includes one of the following:
      - A SELECT statement that references a nickname for a table or view within one of these data sources
      - An open cursor on a nickname for a table or view within one of these data sources
      - Either an INSERT, DELETE, or UPDATE statement issued against a nickname for a table or view within one of these data sources
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
      - NODEGROUP can be specified in place of DATABASE PARTITION GROUP
  - For compatibility with DB2 UDB for OS/390 and z/OS:
      - SYNONYM can be specified in place of ALIAS
      - PROGRAM can be specified in place of PACKAGE

Examples:

Example 1: Drop table TDEPT.

```
DROP TABLE TDEPT
```

Example 2: Drop the view VDEPT.

```
DROP VIEW VDEPT
```

Example 3: The authorization ID HEDGES attempts to drop an alias.

```
DROP ALIAS A1
```

The alias HEDGES.A1 is removed from the catalogs.

Example 4: Hedges attempts to drop an alias, but specifies T1 as the alias-name, where T1 is the name of an existing table (not the name of an alias).

```
DROP ALIAS T1
```

This statement fails (SQLSTATE 42809).

Example 5:

Drop the BUSINESS_OPS database partition group. To drop the database partition group, the two table spaces (ACCOUNTING and PLANS) in the database partition group must first be dropped.

```
DROP TABLESPACE ACCOUNTING
DROP TABLESPACE PLANS
DROP DATABASE PARTITION GROUP BUSINESS_OPS
```

Example 6: Pellow wants to drop the CENTRE function, which he created in his PELLOW schema, using the signature to identify the function instance to be dropped.

```
DROP FUNCTION CENTRE (INT,FLOAT)
```

Example 7: McBride wants to drop the FOCUS92 function, which she created in the PELLOW schema, using the specific name to identify the function instance to be dropped.

```
   DROP SPECIFIC FUNCTION PELLOW.FOCUS92
```

Example 8: Drop the function ATOMIC_WEIGHT from the CHEM schema, where it is known that there is only one function with that name.

```
   DROP FUNCTION CHEM.ATOMIC_WEIGHT
```

Example 9: Drop the trigger SALARY_BONUS, which caused employees under a specified condition to receive a bonus to their salary.

```
   DROP TRIGGER SALARY_BONUS
```

Example 10: Drop the distinct data type named shoesize, if it is not currently in use.

```
   DROP DISTINCT TYPE SHOESIZE
```

Example 11: Drop the SMITHPAY event monitor.

```
   DROP EVENT MONITOR SMITHPAY
```

Example 12: Drop the schema from Example 2 under CREATE SCHEMA using RESTRICT. Notice that the table called PART must be dropped first.

```
  DROP TABLE PART
DROP SCHEMA INVENTRY RESTRICT
```

Example 13: Macdonald wants to drop the DESTROY procedure, which he created in the EIGLER schema, using the specific name to identify the procedure instance to be dropped.

```
   DROP SPECIFIC PROCEDURE  EIGLER.DESTROY
```

Example 14: Drop the procedure OSMOSIS from the BIOLOGY schema, where it is known that there is only one procedure with that name.

```
   DROP PROCEDURE BIOLOGY.OSMOSIS
```

Example 15: User SHAWN used one authorization ID to access the federated database and another to access the database at an Oracle data source called ORACLE1. A mapping was created between the two authorizations, but SHAWN no longer needs to access the data source. Drop the mapping.

```
   DROP USER MAPPING FOR  SHAWN SERVER ORACLE1
```

Example 16: An index of a data source table that a nickname references has been deleted. Drop the index specification that was created to let the optimizer know about this index.

```
   DROP INDEX INDEXSPEC
```

Example 17: Drop the MYSTRUCT1 transform group.

```
   DROP TRANSFORM MYSTRUCT1 FOR POLYGON
```

Example 18: Drop the method BONUS for the EMP data type in the PERSONNEL schema.

```
   DROP METHOD BONUS (SALARY DECIMAL(10,2)) FOR PERSONNEL.EMP
```

Example 19: Drop the sequence ORG_SEQ, with restrictions.

```
   DROP SEQUENCE ORG_SEQ
```

Example 20: A remote table EMPLOYEE was created in a federated system using transparent DDL. Access to the table is no longer needed. Drop the remote table EMPLOYEE.

```
DROP TABLE EMPLOYEE
```

Example 21: Drop the function mapping BONUS_CALC and reinstate the default function mapping (if one exists).

```
DROP FUNCTION MAPPING BONUS_CALC
```

Example 22: Drop the security label component LEVEL.

```
DROP SECURITY LABEL COMPONENT LEVEL
```

Example 23: Drop the security label EMPLOYEESECLABEL of the security policy DATA_ACCESS.

```
DROP SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
```

Example 24: Drop the security policy DATA_ACCESS.

```
DROP SECURITY POLICY DATA_ACCESS
```

Example 25: Drop the security label component GROUPS.

```
DROP SECURITY LABEL COMPONENT GROUPS
```

Example 26: Drop the XML schema EMPLOYEE located in the SQL schema HR.

```
DROP XSROBJECT HR.EMPLOYEE
```

Related concepts:
- "Database authorities" on page 74

Related reference:
- "CREATE FUNCTION MAPPING statement" in *SQL Reference, Volume 2*
- "CREATE TRIGGER statement" in *SQL Reference, Volume 2*
- "CREATE VIEW " on page 1034

Related samples:
- "dbstat.sqb -- Reorganize table and run statistics (MF COBOL)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"
- "tbcreate.sqC -- How to create and drop tables (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)"
- "TbConstr.java -- How to create, use and drop constraints (JDBC)"
- "TbCreate.java -- How to create and drop tables (JDBC)"
- "TbTemp.java -- How to use Declared Temporary Table (JDBC)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "UDFDrop.db2 -- How to uncatalog the Java UDFs contained in UDFsrv.java "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc (C)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbcreate.sqc -- How to create and drop tables (C)"

- "tbtemp.sqc -- How to use a declared temporary table (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "TbConstr.sqlj -- How to create, use and drop constraints (SQLj)"
- "TbCreate.sqlj -- How to create and drop tables (SQLj)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"

GRANT (Database Authorities)

This form of the GRANT statement grants authorities that apply to the entire database (rather than privileges that apply to specific objects within the database).

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

To grant DBADM authority or SECADM authority, SYSADM is required. To grant other authorities, either DBADM or SYSADM authority is required.

Syntax:

```
                 ┌─,─────────────────────────────┐
                 ▼                               │
►►──GRANT────────┬─BINDADD──────────────────┬────┴──ON DATABASE──────────────►
                 ├─CONNECT──────────────────┤
                 ├─CREATETAB────────────────┤
                 ├─CREATE_EXTERNAL_ROUTINE──┤
                 ├─CREATE_NOT_FENCED_ROUTINE┤
                 ├─IMPLICIT_SCHEMA──────────┤
                 ├─DBADM────────────────────┤
                 ├─LOAD─────────────────────┤
                 ├─QUIESCE_CONNECT──────────┤
                 └─SECADM───────────────────┘

         ┌─,───────────────────────────────┐
         ▼                                 │
►──TO────┬─┬─────────┬─authorization-name─┬┴─────────────────────────────►◄
         │ ├─USER────┤                    │
         │ └─GROUP───┘                    │
         └─PUBLIC─────────────────────────┘
```

Description:

BINDADD
: Grants the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if the BINDADD authority is subsequently revoked.

CONNECT
: Grants the authority to access the database.

CREATETAB
: Grants the authority to create base tables. The creator of a base table

automatically has the CONTROL privilege on that table. The creator retains this privilege even if the CREATETAB authority is subsequently revoked.

There is no explicit authority required for view creation. A view can be created at any time if the authorization ID of the statement used to create the view has either CONTROL or SELECT privilege on each base table of the view.

CREATE_EXTERNAL_ROUTINE
Grants the authority to register external routines. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the THREADSAFE clause on the CREATE or ALTER routine statements.)

Once an external routine has been registered, it continues to exist, even if CREATE_EXTERNAL_ROUTINE is subsequently revoked.

CREATE_NOT_FENCED_ROUTINE
Grants the authority to register routines that execute in the database manager's process. Care must be taken that routines so registered will not have adverse side effects. (For more information, see the description of the FENCED clause on the CREATE or ALTER routine statements.)

Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE_NOT_FENCED_ROUTINE is subsequently revoked.

CREATE_EXTERNAL_ROUTINE is automatically granted to an *authorization-name* that is granted CREATE_NOT_FENCED_ROUTINE authority.

IMPLICIT_SCHEMA
Grants the authority to implicitly create a schema.

DBADM
Grants the database administrator authority and all other database authorities except for security administrator authority (SECADM). A database administrator holds nearly all privileges on nearly all objects in the database. The only exceptions are those privileges that are part of the security administrator authority.

A database administrator can grant any privilege that is part of database administrator authority to others.

All database authorities except for SECADM are implicitly and automatically granted to an *authorization-name* that is granted DBADM authority.

LOAD
Grants the authority to load in this database. This authority gives a user the right to use the LOAD utility in this database. SYSADM and DBADM also have this authority by default. However, if a user only has LOAD authority (not SYSADM or DBADM), the user is also required to have table-level privileges. In addition to LOAD privilege, the user is required to have:

- INSERT privilege on the table for LOAD with mode INSERT, TERMINATE (to terminate a previous LOAD INSERT), or RESTART (to restart a previous LOAD INSERT)
- INSERT and DELETE privilege on the table for LOAD with mode REPLACE, TERMINATE (to terminate a previous LOAD REPLACE), or RESTART (to restart a previous LOAD REPLACE)
- INSERT privilege on the exception table, if such a table is used as part of LOAD

QUIESCE_CONNECT
Grants the authority to access the database while it is quiesced.

SECADM
Grants the security administrator authority. The SECADM authority can only be granted to a user. It cannot be granted to a group (SQLSTATE 42521) or to PUBLIC (SQLSTATE 42508). The authority allows the holder to:

- Create and drop security objects such as security labels, security label components, and security policies
- Grant and revoke security labels and exemptions
- Grant and revoke the SETSESSIONUSER privilege
- Execute TRANSFER OWNERSHIP on objects owned by others

No authority other than the SECADM authority is allowed to do these things, not even the SYSADM authority.

TO
Specifies to whom the authorities are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the authorities to all users. DBADM cannot be granted to PUBLIC.

Rules:

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER, or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Notes:

- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - CREATE_NOT_FENCED can be specified in place of CREATE_NOT_FENCED_ROUTINE

Examples:

Example 1: Give the users WINKEN, BLINKEN, and NOD the authority to connect to the database.

```
GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD
```

Example 2: Grant BINDADD authority on the database to a group named D024. There is both a group and a user called D024 in the system.

```
GRANT BINDADD ON DATABASE TO GROUP D024
```

Observe that, the GROUP keyword must be specified; otherwise, an error will occur since both a user and a group named D024 exist. Any member of the D024 group will be allowed to bind packages in the database, but the D024 user will not be allowed (unless this user is also a member of the group D024, had been granted BINDADD authority previously, or BINDADD authority had been granted to another group of which D024 was a member).

Example 3: Give user Walid security administrator authority.

```
GRANT SECADM ON DATABASE TO USER Walid
```

Related reference:

- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

Related samples:

- "dbauth.sqb -- How to grant and display authorities on a database (MF COBOL)"
- "dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)"
- "dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)"
- "DbAuth.java -- Grant, display or revoke privileges on database (JDBC)"
- "DbAuth.sqlj -- Grant, display or revoke privileges on database (SQLj)"

GRANT (Exemption)

This form of the GRANT statement grants to a user an exemption on an access rule for a specified label-based access control (LBAC) security policy. When the user holding the exemption accesses data in a table protected by that security policy the indicated rule will not be enforced when deciding if they can access the data.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
►►──GRANT EXEMPTION ON RULE──┬─access-rule─┬──FOR──policy-name──────────────►
                             └─ALL─────────┘

►──TO USER──authorization-name─────────────────────────────────────────────►◄
```

Description:

EXEMPTION ON RULE
Grants an exemption on an access rule.

access-rule
Identifies the rule for which the exemption is to be granted. The rule must be one of the access rules for the security policy.

ALL
Specifies that exemptions are to be granted for all the access rules for the security policy.

FOR *policy-name*
Identifies the security policy for which the exemption is being granted. The exemption will only be effective for tables that are protected by this security policy. The name must identify a security policy already described in the catalog (SQLSTATE 42704).

TO USER *authorization-name*
Indicates the authorization ID to which the exemption is being granted.

Examples:

Example 1: Grant an exemption on access rule DB2LBACREADSET for security policy DATA_ACCESS to user WALID.

```
GRANT EXEMPTION ON RULE DB2LBACREADSET FOR DATA_ACCESS TO USER WALID
```

Related concepts:
- "Database authorities" on page 74

Related reference:
- "REVOKE (Exemption) " on page 1124

GRANT (Index Privileges)

This form of the GRANT statement grants the CONTROL privilege on indexes.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

GRANT (Index Privileges)

Syntax:

```
►►──GRANT──CONTROL──ON INDEX──index-name──────────────────────────────────────►


        ┌─,────────────────────────────┐
        ▼                              │
►──TO─────┬──┬───────┬──authorization-name─┬──┴───────────────────────────────►◄
          │  ├─USER──┤                    │
          │  └─GROUP─┘                    │
          └─PUBLIC────────────────────────┘
```

Description:

CONTROL
Grants the privilege to drop the index. This is the CONTROL authority for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
Identifies the index for which the CONTROL privilege is to be granted.

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

Rules:

- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Example:

```
GRANT CONTROL ON INDEX DEPTIDX TO USER USER4
```

Related reference:

- "GRANT (Database Authorities) " on page 1081
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096

- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (Package Privileges)

This form of the GRANT statement grants privileges on a package.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- CONTROL privilege on the referenced package
- The WITH GRANT OPTION for each identified privilege on *package-name*
- SYSADM or DBADM authority

SYSADM or DBADM authority is required to grant the CONTROL privilege.

Syntax:

```
              ┌─,──────────┐
              ▼            │
►►──GRANT─────┬─BIND───────┴──────────────────────────────────────────────►
              ├─CONTROL───┤
              │       (1) │
              └─EXECUTE───┘

           (2)
►──ON──PACKAGE──────┬──────────────┬─package-id───────────────────────────►
                    └─schema-name.─┘


         ┌─,─────────────────────────────┐
         ▼                               │
►──TO────┬─┬───────┬─authorization-name─┬┴──┬───────────────────┬──────►◄
         │ ├─USER──┤                    │   └─WITH GRANT OPTION─┘
         │ └─GROUP─┘                    │
         └─PUBLIC───────────────────────┘
```

Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description:

BIND
Grants the privilege to bind a package. The BIND privilege allows a user to

re-issue the BIND command against that package, or to issue the REBIND command. It also allows a user to create a new version of an existing package.

In addition to the BIND privilege, a user must hold the necessary privileges on each table referenced by static DML statements contained in a program. This is necessary, because authorization on static DML statements is checked at bind time.

CONTROL
Grants the privilege to rebind, drop, or execute the package, and extend package privileges to other users. The CONTROL privilege for packages is automatically granted to creators of packages. A package owner is the package binder, or the ID specified with the OWNER option at bind/precompile time.

BIND and EXECUTE are automatically granted to an *authorization-name* that is granted CONTROL privilege.

CONTROL grants the ability to grant the above privileges (except for CONTROL) to others.

EXECUTE
Grants the privilege to execute the package.

ON PACKAGE *schema-name.package-id*
Specifies the name of the package on which privileges are to be granted. If a schema name is not specified, the package ID is implicitly qualified by the default schema. The granting of a package privilege applies to all versions of the package (that is, to all packages that share the same package ID and package schema).

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants the privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name* to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all of the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules:

- If neither USER nor GROUP is specified, then
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.

– If the *authorization-name* is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Notes:

- Package privileges apply to all versions of a package (that is, all packages that share the same package ID and package schema). It is not possible to restrict access to only one version. Because CONTROL privilege is implicitly granted to the binder of a package, if two different users bind two versions of a package, then both users will implicitly be granted access to each other's package.

Examples:

Example 1: Grant the EXECUTE privilege on PACKAGE CORPDATA.PKGA to PUBLIC.

```
GRANT EXECUTE
  ON PACKAGE CORPDATA.PKGA
  TO PUBLIC
```

Example 2: GRANT EXECUTE privilege on package CORPDATA.PKGA to a user named EMPLOYEE. There is neither a group nor a user called EMPLOYEE.

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO EMPLOYEE
```

or

```
GRANT EXECUTE ON PACKAGE
  CORPDATA.PKGA TO USER EMPLOYEE
```

Related reference:

- "GRANT (Database Authorities) " on page 1081
- "GRANT (Index Privileges) " on page 1085
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (Routine Privileges)

This form of the GRANT statement grants privileges on a routine (function, method, or procedure).

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for EXECUTE on the routine
- SYSADM or DBADM authority

To grant all routine EXECUTE privileges in the schema or type, the privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for EXECUTE on all existing and future routines (of the specified type) in the specified schema
- SYSADM or DBADM authority

Syntax:

```
>>──GRANT EXECUTE ON──┬─┤ function-designator ├────────┬──────────────────────────►
                      ├─FUNCTION──┬─────────┬─*────────┤
                      │           └─schema.─┘          │
                      ├─┤ method-designator ├──────────┤
                      ├─METHOD * FOR──┬─type-name────┬─┤
                      │               └─┬─────────┬─*┘ │
                      │                 └─schema.─┘    │
                      ├─┤ procedure-designator ├───────┤
                      └─PROCEDURE──┬─────────┬─*───────┘
                                   └─schema.─┘
```

```
        ┌─,──────────────────────────────────┐
        ▼                                    │
►─TO────┬──┬───────┬──authorization-name─┬───┴─┬───────────────────┬─────────────►◄
        │  ├─USER──┤                     │     └─WITH GRANT OPTION─┘
        │  └─GROUP─┘                     │
        └─PUBLIC─────────────────────────┘
```

Description:

EXECUTE
: Grants the privilege to run the identified user-defined function, method, or procedure.

function-designator
: Uniquely identifies the function.

FUNCTION *schema*.*****
: Identifies all the functions in the schema, including any functions that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

method-designator
: Uniquely identifies the method.

METHOD *
: Identifies all the methods for the type *type-name*, including any methods that may be created in the future.

FOR *type-name*
: Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the

qualifier for unqualified type names. An asterisk (*) can be used in place of *type-name* to identify all types in the schema, including any types that may be created in the future.

procedure-designator
Uniquely identifies the procedure.

PROCEDURE *schema*.*
Identifies all the procedures in the schema, including any procedures that may be created in the future. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

TO
Specifies to whom the EXECUTE privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

PUBLIC
Grants the EXECUTE privilege to all users.

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the EXECUTE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the EXECUTE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to grant the EXECUTE privilege from some other source.

Rules:

- It is not possible to grant the EXECUTE privilege on a function or method defined with schema 'SYSIBM' or 'SYSFUN' (SQLSTATE 42832).
- If neither USER nor GROUP is specified, then:
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both USER and GROUP, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, and no privileges were granted, a warning is returned (SQLSTATE 01007). If the grantor has no privileges on the object of the grant operation, an error is returned (SQLSTATE 42501).

Examples:

Example 1: Grant the EXECUTE privilege on function CALC_SALARY to user JONES. Assume that there is only one function in the schema with function name CALC_SALARY.

```
GRANT EXECUTE ON FUNCTION CALC_SALARY TO JONES
```

Example 2: Grant the EXECUTE privilege on procedure VACATION_ACCR to all users at the current server.

```
GRANT EXECUTE ON PROCEDURE VACATION_ACCR TO PUBLIC
```

Example 3: Grant the EXECUTE privilege on function DEPT_TOTALS to the administrative assistant and give the assistant the ability to grant the EXECUTE privilege on this function to others. The function has the specific name DEPT85_TOT. Assume that the schema has more than one function named DEPT_TOTALS.

```
GRANT EXECUTE ON SPECIFIC FUNCTION DEPT85_TOT
  TO ADMIN_A WITH GRANT OPTION
```

Example 4: Grant the EXECUTE privilege on function NEW_DEPT_HIRES to HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW_DEPT_HIRES.

```
GRANT EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10)) TO HR
```

Example 5: Grant the EXECUTE privilege on method SET_SALARY of type EMPLOYEE to user JONES.

```
GRANT EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE TO JONES
```

Related reference:

- "Function, method, and procedure designators" on page 1286
- "GRANT (Database Authorities) " on page 1081
- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (Schema Privileges)

This form of the GRANT statement grants privileges on a schema.

Invocation:

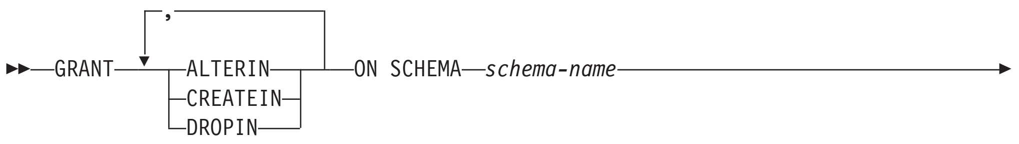
This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

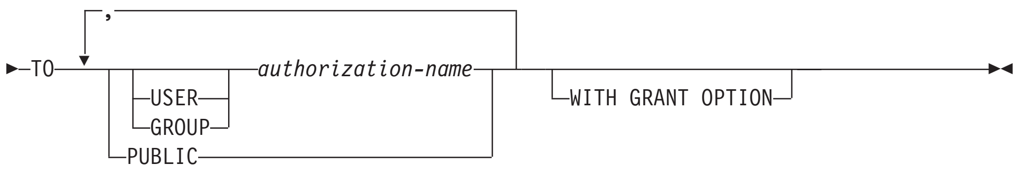
Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- The WITH GRANT OPTION for each identified privilege on *schema-name*
- SYSADM or DBADM authority

No user can grant privileges on any of the following schema names: SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42501).

Syntax:





Description:

ALTERIN
: Grants the privilege to alter or comment on all objects in the schema. The owner of an explicitly created schema automatically receives ALTERIN privilege.

CREATEIN
: Grants the privilege to create objects in the schema. Other authorities or privileges required to create the object (such as CREATETAB) are still required. The owner of an explicitly created schema automatically receives CREATEIN privilege. An implicitly created schema has CREATEIN privilege automatically granted to PUBLIC.

DROPIN
: Grants the privilege to drop all objects in the schema. The owner of an explicitly created schema automatically receives DROPIN privilege.

ON SCHEMA *schema-name*
: Identifies the schema on which the privileges are to be granted.

TO
: Specifies to whom the privileges are granted.

  USER
  : Specifies that the *authorization-name* identifies a user.

  GROUP
  : Specifies that the *authorization-name* identifies a group name.

  authorization-name,...
  : Lists the authorization IDs of one or more users or groups.

    The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

  PUBLIC
  : Grants the privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name*s can only grant the privileges to others if they:
- have DBADM authority or
- received the ability to grant privileges from some other source.

Rules:
- If neither USER nor GROUP is specified, then
  - If the authorization-name is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the authorization-name is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the authorization-name is defined in the operating system as both, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E for MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

Examples:

Example 1: Grant user JSINGLETON to the ability to create objects in schema CORPDATA.

```
GRANT CREATEIN ON SCHEMA CORPDATA TO JSINGLETON
```

Example 2: Grant user IHAKES the ability to create and drop objects in schema CORPDATA.

```
GRANT CREATEIN, DROPIN ON SCHEMA CORPDATA TO IHAKES
```

Related reference:
- "GRANT (Database Authorities) " on page 1081
- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (Security Label)

This form of the GRANT statement grants a label-based access control (LBAC) security label to a user for read access, write access, or for both read and write access.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
>>—GRANT SECURITY LABEL—security-label-name—TO USER—authorization-name——————>

>—┬——————————————————┬———————————————————————————————————————————————————————><
  ├—FOR ALL ACCESS———┤
  ├—FOR READ ACCESS——┤
  └—FOR WRITE ACCESS—┘
```

Description:

SECURITY LABEL *security-label-name*
: Grants the security label *security-label-name*. The name must be qualified with a security policy and must identify a security label already described in the catalog (SQLSTATE 42704).

TO USER *authorization-name*
: Specifies the authorization ID to which the security label is to be granted. The authorization ID must identify a user, not a group or PUBLIC.

FOR ALL ACCESS
: Indicates that the security label is to be granted for both read access and write access.

FOR READ ACCESS
: Indicates that the security label is to be granted for read access only.

FOR WRITE ACCESS
: Indicates that the security label is to be granted for write access only.

Rules:

- For any given security policy a user can be granted at most one security label from that policy for read access and one for write access. If the grantee already holds a security label for the type of access (read or write) indicated and that is part of the security policy that qualifies *security-label-name*, an error is returned (SQLSTATE 428GR).
- If a user holds different security labels for read access and write access, the security labels must meet the following criteria (SQLSTATE 428GQ):
  - If any component in the security labels is of type ARRAY then the value for that component must be the same in both security labels.
  - If any component in the security labels is of type SET then every element in the value for that component in the write security label must also be part of the value for that component in the read security label.
  - If any component in the security labels is of type TREE then every element in the value for that component in the write security label must be the same as or a descendent of one of the elements in the value for that same component in the read security label.

Examples:

Example 1: The following statement grants two security labels to user GUYLAINE. The security label EMPLOYEESECLABELREAD is granted for read access and the security label EMPLOYEESECLABELWRITE is granted for write access. Both security labels are part of the security policy DATA_ACCESS.

```
GRANT SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELREAD
  TO USER GUYLAINE FOR READ ACCESS

GRANT SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABELWRITE
  TO USER GUYLAINE FOR WRITE ACCESS
```

The same user is now granted the security label BEGINNER for both read and write access. This does not cause an error, because BEGINNER is part of the security policy CLASSPOLICY, and the security labels already held are part of the security policy DATA_ACCESS.

```
GRANT SECURITY LABEL CLASSPOLICY.BEGINNER
  TO USER GUYLAINE FOR ALL ACCESS
```

Related concepts:

- "Database authorities" on page 74

Related reference:

- "REVOKE (Security Label) " on page 1133

GRANT (Sequence Privileges)

This form of the GRANT statement grants privileges on a sequence.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for each identified privilege on *sequence-name*
- SYSADM or DBADM authority

Syntax:

```
              ┌─,─────────┐
              ▼           │
►►──GRANT─────┬─USAGE─┬───┴──ON SEQUENCE──sequence-name──────────────►
              └─ALTER─┘
```

```
          ┌─,──────────────────────────────┐
          ▼                                │
►─TO──────┬──────────┬─authorization-name─┬┴──┬───────────────────┬──────────►◄
          │ ├─USER──┤                     │   └─WITH GRANT OPTION─┘
          │ └─GROUP─┘                     │
          └─PUBLIC────────────────────────┘
```

Description:

USAGE
Grants the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

ALTER
Grants the privilege to alter sequence properties using the ALTER SEQUENCE statement.

ON SEQUENCE *sequence-name*
Identifies the sequence on which the specified privileges are to be granted. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error (SQLSTATE 42704) is returned.

TO
Specifies to whom the specified privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

PUBLIC
Grants the specified privileges to all users.

WITH GRANT OPTION
Allows the specified *authorization-name* to grant the specified privileges to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only grant the specified privileges to others if they:
- have SYSADM or DBADM authority or
- received the ability to grant the specified privileges from some other source.

Rules:
- If neither USER nor GROUP is specified, then:
  – If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  – If the *authorization-name* is defined in the operating system only as USER or if it is undefined, then USER is assumed.
  – If the *authorization-name* is defined in the operating system as both USER and GROUP, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges is not granted. If no privileges are granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or

MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.)

Example:

Example 1: Grant any user the USAGE privilege on a sequence called ORG_SEQ.

```
GRANT USAGE ON SEQUENCE ORG_SEQ TO PUBLIC
```

Example 2: Grant user BOBBY the ability to alter a sequence called GENERATE_ID, and to grant this privilege to others.

```
GRANT ALTER ON SEQUENCE GENERATE_ID TO BOBBY WITH GRANT OPTION
```

Related reference:

- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Database Authorities) " on page 1081
- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (Server Privileges)

This form of the GRANT statement grants the privilege to access and use a specified data source in pass-through mode.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

Syntax:

```
►►──GRANT PASSTHRU ON SERVER──server-name──TO──────────────────────────►


    ┌─,─────────────────────────────────┐
    ▼                                   │
►───┬─┬───────┬──authorization-name─┬───┴──────────────────────────────►◄
    │ ├─USER──┤                     │
    │ └─GROUP─┘                     │
    └─PUBLIC────────────────────────┘
```

Description:

server-name
Names the data source for which the privilege to use in pass-through mode is being granted. *server-name* must identify a data source that is described in the catalog.

TO
Specifies to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Grants to all users the privilege to pass through to *server-name.*

Examples:

Example 1: Give R. Smith and J. Jones the privilege to pass through to data source SERVALL. Their authorization IDs are RSMITH and JJONES.

```
   GRANT PASSTHRU ON SERVER SERVALL
TO USER RSMITH,
USER JJONES
```

Example 2: Grant the privilege to pass through to data source EASTWING to a group whose authorization ID is D024. There is a user whose authorization ID is also D024.

```
   GRANT PASSTHRU ON SERVER EASTWING TO GROUP D024
```

The GROUP keyword must be specified; otherwise, an error will occur because D024 is a user's ID as well as the specified group's ID (SQLSTATE 56092). Any member of group D024 will be allowed to pass through to EASTWING. Therefore, if user D024 belongs to the group, this user will be able to pass through to EASTWING.

Related reference:
- "GRANT (Database Authorities) " on page 1081
- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Table Space Privileges) " on page 1101
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (SETSESSIONUSER Privilege)

This form of the GRANT statement grants the SETSESSIONUSER privilege to one or more authorization IDs. The privilege allows the holder to use the SET SESSION AUTHORIZATION statement to set the session authorization to one of a set of specified authorization IDs.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
                              ┌─,──────────────────────────────────┐      ┌─,─────────────────────────────┐
                              ▼                                    │      ▼                               │
►►──GRANT SETSESSIONUSER ON────┬─USER──session-authorization-name─┬┴──TO────┬─USER──┬──authorization-name─┴──────────►◄
                               └─PUBLIC───────────────────────────┘         └─GROUP─┘
```

Description:

SETSESSIONUSER ON
Grants the privilege to assume the identity of a new authorization ID.

USER *session-authorization-name*
Specifies the authorization ID that the *authorization-name* will be able to assume, using the SET SESSION AUTHORIZATION statement. The *session-authorization-name* must identify a user, not a group.

PUBLIC
Specifies that the grantee will be able to assume any valid authorization ID, using the SET SESSION AUTHORIZATION statement.

TO
Specifies to whom the privilege is granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

authorization-name,...
Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

Examples:

Example 1: The following statement grants user PAUL the ability to set the session authorization to user WALID and therefore to execute statements as WALID.

```
GRANT SETSESSIONUSER ON USER WALID
  TO USER PAUL
```

Example 2: The following statement grants user GUYLAINE the ability to set the session authorization to user BOBBY. It also grants her the ability to set the session authorization to users RICK and KEVIN.

```
GRANT SETSESSIONUSER ON USER BOBBY, USER RICK, USER KEVIN
  TO USER GUYLAINE
```

Example 3: The following statement grants user WALID and everyone in the groups ADMINS and ACCTG the ability to set the session authorization to any user.

```
GRANT SETSESSIONUSER ON PUBLIC TO USER WALID, GROUP ADMINS, ACCTG
```

Related concepts:

- "Database authorities" on page 74

Related reference:

- "SET SESSION AUTHORIZATION statement" in *SQL Reference, Volume 2*
- "REVOKE (SETSESSIONUSER Privilege) " on page 1138

GRANT (Table Space Privileges)

This form of the GRANT statement grants privileges on a table space.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- The WITH GRANT OPTION for use of the table space
- SYSADM, SYSCTRL, or DBADM authority

Syntax:

```
►►──GRANT──USE──OF TABLESPACE──tablespace-name──TO─────────────────────────►

    ┌─,──────────────────────────────┐
    ▼                                │
►────┬─┬───────┬─authorization-name─┴──┬───────────────────┬──────────────►◄
     │ ├─USER──┤                        └─WITH GRANT OPTION─┘
     │ └─GROUP─┘
     └─PUBLIC────────────────────────┘
```

Description:

USE
: Grants the privilege to specify or default to the table space when creating a table. The creator of a table space automatically receives USE privilege with grant option.

OF TABLESPACE *tablespace-name*
: Identifies the table space on which the USE privilege is to be granted. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a system temporary table space (SQLSTATE 42809).

TO
: Specifies to whom the USE privilege is granted.

USER
: Specifies that the *authorization-name* identifies a user.

GROUP
: Specifies that the *authorization-name* identifies a group name.

authorization-name
: Lists the authorization IDs of one or more users or groups.

The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
: Grants the USE privilege to all users.

WITH GRANT OPTION
: Allows the specified *authorization-name* to GRANT the USE privilege to others.

If the WITH GRANT OPTION is omitted, the specified *authorization-name* can only GRANT the USE privilege to others if they:

- have SYSADM or DBADM authority or
- received the ability to GRANT the USE privilege from some other source.

Notes:

If neither USER nor GROUP is specified, then

- If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
- If the *authorization-name* is defined in the operating system only as USER, or if it is undefined, then USER is assumed.
- If the *authorization-name* is defined in the operating system as both, an error (SQLSTATE 56092) is returned.

Examples:

Example 1: Grant user BOBBY the ability to create tables in table space PLANS and to grant this privilege to others.

```
GRANT USE OF TABLESPACE PLANS TO BOBBY WITH GRANT OPTION
```

Related reference:

- "GRANT (Database Authorities) " on page 1081
- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table, View, or Nickname Privileges) " on page 1103

GRANT (Table, View, or Nickname Privileges)

This form of the GRANT statement grants privileges on a table, view, or nickname.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

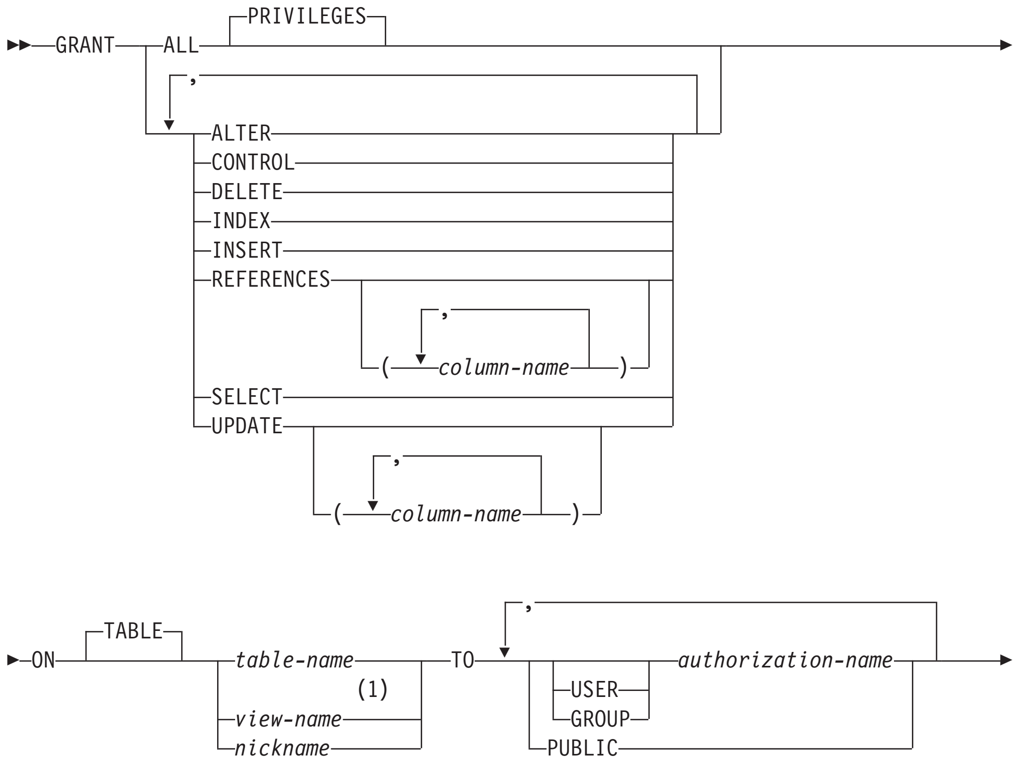
Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced table, view, or nickname
- The WITH GRANT OPTION for each identified privilege. If ALL is specified, the authorization ID must have some grantable privilege on the identified table, view, or nickname.
- SYSADM or DBADM authority

DBADM or SYSADM authority is required to grant the CONTROL privilege, or to grant privileges on catalog tables and views.

Syntax:



```
►─┬───────────────────┬─────────────────────────────────────────────────►◄
  └─WITH GRANT OPTION─┘
```

Notes:

1 ALTER, INDEX, and REFERENCES privileges are not applicable to views.

Description:

ALL or **ALL PRIVILEGES**
Grants all the appropriate privileges, except CONTROL, on the base table, view, or nickname named in the ON clause.

If the authorization ID of the statement has CONTROL privilege on the table, view, or nickname, or DBADM or SYSADM authority, then all the privileges applicable to the object (except CONTROL) are granted. Otherwise, the privileges granted are all those grantable privileges that the authorization ID of the statement has on the identified table, view, or nickname.

If ALL is not specified, one or more of the keywords in the list of privileges must be specified.

ALTER
Grants the privilege to:
- Add columns to a base table definition.
- Create or drop a primary key or unique constraint on a base table.
- Create or drop a foreign key on a base table.

  The REFERENCES privilege on each column of the parent table is also required.
- Create or drop a check constraint on a base table.
- Create a trigger on a base table.
- Add, reset, or drop a column option for a nickname.
- Change a nickname column name or data type.
- Add or change a comment on a base table or a nickname.

CONTROL
Grants:
- All of the appropriate privileges in the list, that is:
  - ALTER, CONTROL, DELETE, INSERT, INDEX, REFERENCES, SELECT, and UPDATE to base tables
  - CONTROL, DELETE, INSERT, SELECT, and UPDATE to views
  - ALTER, CONTROL, INDEX, and REFERENCES to nicknames
- The ability to grant the above privileges (except for CONTROL) to others.
- The ability to drop the base table, view, or nickname.

  This ability cannot be extended to others on the basis of holding CONTROL privilege. The only way that it can be extended is by granting the CONTROL privilege itself and that can only be done by someone with SYSADM or DBADM authority.
- The ability to execute the RUNSTATS utility on the table and indexes.
- The ability to execute the REORG utility on the table.
- The ability to issue the SET INTEGRITY statement against a base table, materialized query table, or staging table.

The definer of a base table, materialized query table, staging table, or nickname automatically receives the CONTROL privilege.

The definer of a view automatically receives the CONTROL privilege if the definer holds the CONTROL privilege on all tables, views, and nicknames identified in the fullselect.

DELETE
Grants the privilege to delete rows from the table or updatable view.

INDEX
Grants the privilege to create an index on a table, or an index specification on a nickname. This privilege cannot be granted on a view. The creator of an index or index specification automatically has the CONTROL privilege on the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains the CONTROL privilege even if the INDEX privilege is revoked.

INSERT
Grants the privilege to insert rows into the table or updatable view and to run the IMPORT utility.

REFERENCES
Grants the privilege to create and drop a foreign key referencing the table as the parent.

If the authorization ID of the statement has one of:

- DBADM or SYSADM authority
- CONTROL privilege on the table
- REFERENCES WITH GRANT OPTION on the table

then the grantee(s) can create referential constraints using all columns of the table as parent key, even those added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column REFERENCES privileges that the authorization ID of the statement has on the identified table.

The privilege can be granted on a nickname, although foreign keys cannot be defined to reference nicknames.

REFERENCES (*column-name***,...)**
Grants the privilege to create and drop a foreign key using only those columns specified in the column list as a parent key. Each *column-name* must be an unqualified name that identifies a column of the table identified in the ON clause. Column level REFERENCES privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

SELECT
Grants the privilege to:

- Retrieve rows from the table or view.
- Create views on the table.
- Run the EXPORT utility against the table or view.

UPDATE
Grants the privilege to use the UPDATE statement on the table or updatable view identified in the ON clause.

If the authorization ID of the statement has one of:

- DBADM or SYSADM authority
- CONTROL privilege on the table or view

- UPDATE WITH GRANT OPTION on the table or view

then the grantee(s) can update all updatable columns of the table or view on which the grantor has with grant privilege as well as those columns added later using the ALTER TABLE statement. Otherwise, the privileges granted are all those grantable column UPDATE privileges that the authorization ID of the statement has on the identified table or view.

UPDATE (*column-name***,...)**
Grants the privilege to use the UPDATE statement to update only those columns specified in the column list. Each *column-name* must be an unqualified name that identifies a column of the table or view identified in the ON clause. Column level UPDATE privilege cannot be granted on typed tables, typed views, or nicknames (SQLSTATE 42997).

ON TABLE *table-name* or *view-name* or *nickname*
Specifies the table, view, or nickname on which privileges are to be granted.

No privileges may be granted on an inoperative view or an inoperative materialized query table (SQLSTATE 51024). No privileges may be granted on a declared temporary table (SQLSTATE 42995).

TO
Specifies to whom the privileges are granted.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups. (Previous restrictions on grants to the authorization ID of the user issuing the statement have been removed.)

A privilege that is granted to a group is not used for authorization checking:
- On static DML statements in a package
- On a base table while processing a CREATE VIEW statement
- On a base table while processing a CREATE TABLE statement for a materialized query table

In DB2 Database for Linux, UNIX, and Windows, table privileges granted to groups only apply to statements that are dynamically prepared. For example, if the INSERT privilege on the PROJECT table has been granted to group D204 but not UBIQUITY (a member of D204) UBIQUITY could issue the statement:

```
EXEC SQL EXECUTE IMMEDIATE :INSERT_STRING;
```

where the content of the string is:

```
INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

but could not precompile or bind a program with the statement:

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
VALUES ('AD3114', 'TOOL PROGRAMMING', 'D21', '000260');
```

PUBLIC
Grants the privileges to all users. (Previous restrictions on the use of privileges granted to PUBLIC for static SQL statements and the CREATE VIEW statement have been removed.)

WITH GRANT OPTION
Allows the specified *authorization-name*s to GRANT the privileges to others.

If the specified privileges include CONTROL, the WITH GRANT OPTION applies to all the applicable privileges except for CONTROL (SQLSTATE 01516).

Rules:

- If neither USER nor GROUP is specified, then
  - If the *authorization-name* is defined in the operating system only as GROUP, then GROUP is assumed.
  - If the *authorization-name* is defined in the operating system only as USER or if it is undefined, USER is assumed.
  - If the *authorization-name* is defined in the operating system as both, an error (SQLSTATE 56092) is returned.
- In general, the GRANT statement will process the granting of privileges that the authorization ID of the statement is allowed to grant, returning a warning (SQLSTATE 01007) if one or more privileges was not granted. If no privileges were granted, an error is returned (SQLSTATE 42501). (If the package used for processing the statement was precompiled with LANGLEVEL set to SQL92E or MIA, a warning is returned (SQLSTATE 01007), unless the grantor has no privileges on the object of the grant operation.) If CONTROL privilege is specified, privileges will only be granted if the authorization ID of the statement has SYSADM or DBADM authority (SQLSTATE 42501).

Notes:

- Privileges may be granted independently at every level of a table hierarchy. A user with a privilege on a supertable may affect the subtables. For example, an update specifying the supertable *T* may show up as a change to a row in the subtable *S* of *T* done by a user with UPDATE privilege on *T* but without UPDATE privilege on *S*. A user can only operate directly on the subtable if the necessary privilege is held on the subtable.
- Granting nickname privileges has no effect on data source object (table or view) privileges. Typically, data source privileges are required for the table or view that a nickname references when attempting to retrieve data.
- ***Compatibilities***
  - For compatibility with DB2 UDB for OS/390 and z/OS:
    - The following syntax is tolerated and ignored:
      - PUBLIC AT ALL LOCATIONS

Examples:

Example 1: Grant all privileges on the table WESTERN_CR to PUBLIC.

```
GRANT ALL ON WESTERN_CR
   TO PUBLIC
```

Example 2: Grant the appropriate privileges on the CALENDAR table so that users PHIL and CLAIRE can read it and insert new entries into it. Do not allow them to change or remove any existing entries.

```
GRANT SELECT, INSERT ON CALENDAR
   TO USER  PHIL, USER CLAIRE
```

Example 3: Grant all privileges on the COUNCIL table to user FRANK and the ability to extend all privileges to others.

```
GRANT ALL ON COUNCIL
   TO USER FRANK WITH GRANT OPTION
```

Example 4: GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a user named JOHN. There is a user called JOHN and no group called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

or

```
GRANT SELECT
  ON CORPDATA.EMPLOYEE TO USER JOHN
```

Example 5: GRANT SELECT privilege on table CORPDATA.EMPLOYEE to a group named JOHN. There is a group called JOHN and no user called JOHN.

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO JOHN
```

or

```
GRANT SELECT ON CORPDATA.EMPLOYEE TO GROUP JOHN
```

Example 6: GRANT INSERT and SELECT on table T1 to both a group named D024 and a user named D024.

```
GRANT INSERT, SELECT ON TABLE T1
  TO GROUP D024, USER D024
```

In this case, both the members of the D024 group and the user D024 would be allowed to INSERT into and SELECT from the table T1. Also, there would be two rows added to the SYSCAT.TABAUTH catalog view.

Example 7: GRANT INSERT, SELECT, and CONTROL on the CALENDAR table to user FRANK. FRANK must be able to pass the privileges on to others.

```
GRANT CONTROL ON TABLE CALENDAR
  TO FRANK WITH GRANT OPTION
```

The result of this statement is a warning (SQLSTATE 01516) that CONTROL was not given the WITH GRANT OPTION. Frank now has the ability to grant any privilege on CALENDAR including INSERT and SELECT as required. FRANK cannot grant CONTROL on CALENDAR to other users unless he has SYSADM or DBADM authority.

Example 8: User JON created a nickname for an Oracle table that had no index. The nickname is ORAREM1. Later, the Oracle DBA defined an index for this table. User SHAWN now wants DB2 to know that this index exists, so that the optimizer can devise strategies to access the table more efficiently. SHAWN can inform DB2 of the index by creating an index specification for ORAREM1. Give SHAWN the index privilege on this nickname, so that he can create the index specification.

```
GRANT INDEX ON NICKNAME ORAREM1
  TO USER SHAWN
```

Related reference:

- “ALTER TABLE ” on page 854
- “GRANT (Database Authorities) ” on page 1081

- "GRANT (Index Privileges) " on page 1085
- "GRANT (Package Privileges) " on page 1087
- "GRANT (Routine Privileges) " on page 1089
- "GRANT (Schema Privileges) " on page 1092
- "GRANT (Sequence Privileges) " on page 1096
- "GRANT (Server Privileges) " on page 1098
- "GRANT (Table Space Privileges) " on page 1101

Related samples:
- "tbpriv.sqc -- How to grant, display, and revoke privileges (C)"
- "tbpriv.sqC -- How to grant, display, and revoke privileges (C++)"
- "TbPriv.java -- How to grant, display and revoke privileges on a table (JDBC)"
- "TbPriv.sqlj -- How to grant, display and revoke privileges on a table (SQLj)"

INSERT

The INSERT statement inserts rows into a table, nickname, or view, or the underlying tables, nicknames, or views of the specified fullselect. Inserting a row into a nickname inserts the row into the data source object to which the nickname refers. Inserting a row into a view also inserts the row into the table on which the view is based, if no INSTEAD OF trigger is defined for the insert operation on this view. If such a trigger is defined, the trigger will be executed instead.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- INSERT privilege on the target table, view, or nickname
- CONTROL privilege on the target table, view, or nickname
- SYSADM or DBADM authority

In addition, for each table, view, or nickname referenced in any fullselect used in the INSERT statement, the privileges held by the authorization ID of the statement must include at least one of the following:
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority

GROUP privileges are not checked for static INSERT statements.

If the target of the insert operation is a nickname, the privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax:

```
>>──INSERT INTO──┬─table-name──────┬──┬──────────────────────────┬──────────────>
                 ├─view-name───────┤  │    ┌─,───────────┐       │
                 ├─nickname────────┤  │    ▼             │       │
                 └─(──fullselect──)┘  └─(──┴─column-name─┴───)──┘

>──┬───────────────────┬───────────────────────────────────────────────────────>
   └─┤ include-columns ├─┘

            ┌─,────────────────────────────────┐
            ▼                                  │
>──┬─VALUES───┬─┬─expression─┬─────────────────┬─┴──┬──┬────────────────┬──────><
   │          │ ├─NULL───────┤                 │    │  └─WITH──┬─RR─┬───┘
   │          │ └─DEFAULT────┘                 │    │          ├─RS─┤
   │          │      ┌─,───────────────┐       │    │          ├─CS─┤
   │          │      ▼                 │       │    │          └─UR─┘
   │          └─(────┬─expression─┬────┴──)────┘    │
   │                 ├─NULL───────┤                 │
   │                 └─DEFAULT────┘                 │
   └─┬───────────────────────────────────────┬─fullselect─┘
     │        ┌─,────────────────────────┐   │
     │        ▼                          │   │
     └─WITH───┴─common-table-expression──┴───┘
```

include-columns:

```
              ┌─,──────────────────────────┐
              ▼                            │
├──INCLUDE──(──┴─column-name──data-type──┴──)────────────────────────────────┤
```

Description:

INTO *table-name*, *view-name*, *nickname*, **or** *(fullselect)*

Identifies the object of the insert operation. The name must identify a table, view or nickname that exists at the application server, but it must not identify a catalog table, a system-maintained materialized query table, a view of a catalog table, or a read-only view, unless an INSTEAD OF trigger is defined for the insert operation on the subject view. Rows inserted into a nickname are placed in the data source object to which the nickname refers.

If the object of the insert operation is a fullselect, the fullselect must be insertable, as defined in the "Insertable views" Notes item in the description of the CREATE VIEW statement.

If no INSTEAD OF trigger exists for the insert operation on this view, a value cannot be inserted into a view column that is derived from:

- A constant, expression, or scalar function
- The same base table column as some other column of the view

If the object of the insert operation is a view with such columns, a list of column names must be specified, and the list must not identify these columns.

A row can be inserted into a view or a fullselect that is defined using a UNION ALL if the row satisfies the check constraints of exactly one of the underlying base tables. If a row satisfies the check constraints of more than one table, or no table at all, an error is returned (SQLSTATE 23513).

(*column-name***,...)**
Specifies the columns for which insert values are provided. Each name must identify a column of the specified table, view, or nickname, or a column in the fullselect. The same column must not be identified more than once. A column that cannot accept inserted values (for example, a column based on an expression) must not be identified.

Omission of the column list is an implicit specification of a list in which every column of the table or view, or every item in the select-list of the fullselect is identified in left-to-right order. This list is established when the statement is prepared and, therefore, does not include columns that were added to a table after the statement was prepared.

include-columns
Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the INSERT statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the INSERT statement. This clause can only be specified if the INSERT statement is nested in the FROM clause of a fullselect.

column-name
Specifies a column of the intermediate result table of the INSERT statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

VALUES
Introduces one or more rows of values to be inserted.

Each host variable named must be described in the program in accordance with the rules for declaring host variables.

The number of values for each row must equal the number of names in the implicit or explicit column list and the columns identified in the INCLUDE clause. The first value is inserted in the first column in the list, the second value in the second column, and so on.

expression
An *expression* can be any expression defined in “Expressions”.

NULL
Specifies the null value and should only be specified for nullable columns.

DEFAULT
Specifies that the default value is to be used. The result of specifying DEFAULT depends on how the column was defined, as follows:

- If the column was defined as a generated column based on an expression, the column value is generated by the system, based on that expression.
- If the IDENTITY clause is used, the value is generated by the database manager.
- If the WITH DEFAULT clause is used, the value inserted is as defined for the column (see *default-clause* in "CREATE TABLE").
- If the NOT NULL clause is used and the GENERATED clause is not used, or the WITH DEFAULT clause is not used or DEFAULT NULL is used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).
- When inserting into a nickname, the DEFAULT keyword will be passed through the INSERT statement to the data source only if the data source supports the DEFAULT keyword in its query language syntax.

WITH *common-table-expression*
Defines a common table expression for use with the fullselect that follows.

fullselect
Specifies a set of new rows in the form of the result table of a fullselect. There may be one, more than one, or none. If the result table is empty, SQLCODE is set to +100 and SQLSTATE is set to '02000'.

When the base object of the INSERT and the base object of the fullselect or any subquery of the fullselect, are the same table, the fullselect is completely evaluated before any rows are inserted.

The number of columns in the result table must equal the number of names in the column list. The value of the first column of the result is inserted in the first column in the list, the second value in the second column, and so on.

WITH
Specifies the isolation level at which the fullselect is executed.

RR
Repeatable Read

RS
Read Stability

CS
Cursor Stability

UR
Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules:

- ***Triggers:*** INSERT statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the inserted values. If an insert operation into a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- ***Default values:*** The value inserted in any column that is not in the column list is either the default value of the column or null. Columns that do not allow null

values and are not defined with NOT NULL WITH DEFAULT must be included in the column list. Similarly, if you insert into a view, the value inserted into any column of the base table that is not in the view is either the default value of the column or null. Hence, all columns of the base table that are not in the view must have either a default value or allow null values. The only value that can be inserted into a generated column defined with the GENERATED ALWAYS clause is DEFAULT (SQLSTATE 428C9).

- ***Length:*** If the insert value of a column is a number, the column must be a numeric column with the capacity to represent the integral part of the number. If the insert value of a column is a string, the column must either be a string column with a length attribute at least as great as the length of the string, or a datetime column if the string represents a date, time, or timestamp.
- ***Assignment:*** Insert values are assigned to columns in accordance with specific assignment rules.
- ***Validity:*** If the table named, or the base table of the view named, has one or more unique indexes, each row inserted into the table must conform to the constraints imposed by those indexes. If a view whose definition includes WITH CHECK OPTION is named, each row inserted into the view must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW".
- ***Referential Integrity:*** For each constraint defined on a table, each non-null insert value of the foreign key must be equal to a primary key value of the parent table.
- ***Check Constraint:*** Insert values must satisfy the check conditions of the check constraints defined on the table. An INSERT to a table with check constraints defined has the constraint conditions evaluated once for each row that is inserted.
- ***XML values:*** A value that is inserted into an XML column must be a well-formed XML document (SQLSTATE 2200M).

Notes:

- After execution of an INSERT statement, the value of the third variable of the SQLERRD(3) portion of the SQLCA indicates the number of rows that were passed to the insert operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. SQLERRD(5) contains the count of all triggered insert, update and delete operations.
- Unless appropriate locks already exist, one or more exclusive locks are acquired at the execution of a successful INSERT statement. Until the locks are released, an inserted row can only be accessed by:
  - The application process that performed the insert.
  - Another application process using isolation level UR through a read-only cursor, SELECT INTO statement, or subselect used in a subquery.
- For further information about locking, see the description of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- If an application is running against a partitioned database, and it is bound with option INSERT BUF, then INSERT with VALUES statements which are not processed using EXECUTE IMMEDIATE may be buffered. DB2 assumes that such an INSERT statement is being processed inside a loop in the application's logic. Rather than execute the statement to completion, it attempts to buffer the new row values in one or more buffers. As a result the actual insertions of the rows into the table are performed later, asynchronous with the application's

INSERT logic. Be aware that this asynchronous insertion may cause an error related to an INSERT to be returned on some other SQL statement that follows the INSERT in the application.

This has the potential to dramatically improve INSERT performance, but is best used with clean data, due to the asynchronous nature of the error handling.

- When a row is inserted into a table that has an identity column, DB2 generates a value for the identity column.
  - For a GENERATED ALWAYS identity column, DB2 always generates the value.
  - For a GENERATED BY DEFAULT column, if a value is not explicitly specified (with a VALUES clause, or subselect), DB2 generates a value.

  The first value generated by DB2 is the value of the START WITH specification for the identity column.
- When a value is inserted for a user-defined distinct type identity column, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type prior to the computation.)
- When inserting into a GENERATED ALWAYS identity column, DB2 will always generate a value for the column, and users must not specify a value at insertion time. If a GENERATED ALWAYS identity column is listed in the column-list of the INSERT statement, with a non-DEFAULT value in the VALUES clause, an error occurs (SQLSTATE 428C9).

  For example, assuming that EMPID is defined as an identity column that is GENERATED ALWAYS, then the command:

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (:hv_valid_emp_id, :hv_name, :hv_addr)
  ```

  will result in an error.
- When inserting into a GENERATED BY DEFAULT column, DB2 will allow an actual value for the column to be specified within the VALUES clause, or from a subselect. However, when a value is specified in the VALUES clause, DB2 does not perform any verification of the value. In order to guarantee uniqueness of the values, a unique index on the identity column must be created.

  When inserting into a table with a GENERATED BY DEFAULT identity column, without specifying a column list, the VALUES clause can specify the DEFAULT keyword to represent the value for the identity column. DB2 will generate the value for the identity column.

  ```
  INSERT INTO T2 (EMPID, EMPNAME, EMPADDR)
    VALUES (DEFAULT, :hv_name, :hv_addr)
  ```

  In this example, EMPID is defined as an identity column, and thus the value inserted into this column is generated by DB2.
- The rules for inserting into an identity column with a subselect are similar to those for an insert with a VALUES clause. A value for an identity column may only be specified if the identity column is defined as GENERATED BY DEFAULT.

  For example, assume T1 and T2 are tables with the same definition, both containing columns *intcol1* and *identcol2* (both are type INTEGER and the second column has the identity attribute). Consider the following insert:

  ```
  INSERT INTO T2
    SELECT *
    FROM T1
  ```

This example is logically equivalent to:

```
INSERT INTO T2 (intcol1,identcol2)
  SELECT intcol1, identcol2
  FROM T1
```

In both cases, the INSERT statement is providing an explicit value for the identity column of T2. This explicit specification can be given a value for the identity column, but the identity column in T2 must be defined as GENERATED BY DEFAULT. Otherwise, an error will result (SQLSTATE 428C9).

If there is a table with a column defined as a GENERATED ALWAYS identity, it is still possible to propagate all other columns from a table with the same definition. For example, given the example tables T1 and T2 described above, the intcol1 values from T1 to T2 can be propagated with the following SQL:

```
INSERT INTO T2 (intcol1)
  SELECT intcol1
  FROM T1
```

Note that, because identcol2 is not specified in the column-list, it will be filled in with its default (generated) value.

- When inserting a row into a single column table where the column is defined as a GENERATED ALWAYS identity column, it is possible to specify a VALUES clause with the DEFAULT keyword. In this case, the application does not provide any value for the table, and DB2 generates the value for the identity column.

  ```
  INSERT INTO IDTABLE
    VALUES(DEFAULT)
  ```

  Assuming the same single column table for which the column has the identity attribute, to insert multiple rows with a single INSERT statement, the following INSERT statement could be used:

  ```
  INSERT INTO IDTABLE
    VALUES (DEFAULT), (DEFAULT), (DEFAULT), (DEFAULT)
  ```
- When DB2 generates a value for an identity column, that generated value is consumed; the next time that a value is needed, DB2 will generate a new value. This is true even when an INSERT statement involving an identity column fails or is rolled back.

  For example, assume that a unique index has been created on the identity column. If a duplicate key violation is detected in generating a value for an identity column, an error occurs (SQLSTATE 23505) and the value generated for the identity column is considered to be consumed. This can occur when the identity column is defined as GENERATED BY DEFAULT and the system tries to generate a new value, but the user has explicitly specified values for the identity column in previous INSERT statements. Reissuing the same INSERT statement in this case can lead to success. DB2 will generate the next value for the identity column, and it is possible that this next value will be unique, and that this INSERT statement will be successful.
- If the maximum value for the identity column is exceeded (or minimum value for a descending sequence) in generating a value for an identity column, an error occurs (SQLSTATE 23522). In this situation, the user would have to DROP and CREATE a new table with an identity column having a larger range (that is, change the data type or increment value for the column to allow for a larger range of values).

  For example, an identity column may have been defined with a data type of SMALLINT, and eventually the column runs out of assignable values. To redefine the identity column as INTEGER, the data would need to be unloaded, the table would have to be dropped and recreated with a new definition for the column, and then the data would be reloaded. When the table is redefined, it needs to specify a START WITH value for the identity column such that the next value generated by DB2 will be the next value in the original sequence. To determine the end value, issue a query using MAX of the identity column (for an ascending sequence), or MIN of the identity column (for a descending sequence), before unloading the data.

Examples:

Example 1: Insert a new department with the following specifications into the DEPARTMENT table:

- Department number (DEPTNO) is 'E31'
- Department name (DEPTNAME) is 'ARCHITECTURE'
- Managed by (MGRNO) a person with number '00390'
- Reports to (ADMRDEPT) department 'E01'.

```
INSERT INTO DEPARTMENT
  VALUES ('E31', 'ARCHITECTURE', '00390', 'E01')
```

Example 2: Insert a new department into the DEPARTMENT table as in example 1, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT )
  VALUES ('E31', 'ARCHITECTURE', 'E01')
```

Example 3: Insert two new departments using one statement into the DEPARTMENT table as in example 2, but do not assign a manager to the new department.

```
INSERT INTO DEPARTMENT (DEPTNO, DEPTNAME, ADMRDEPT)
  VALUES ('B11', 'PURCHASING', 'B01'),
         ('E41', 'DATABASE ADMINISTRATION', 'E01')
```

Example 4: Create a temporary table MA_EMP_ACT with the same columns as the EMP_ACT table. Load MA_EMP_ACT with the rows from the EMP_ACT table with a project number (PROJNO) starting with the letters 'MA'.

```
CREATE TABLE MA_EMP_ACT
         ( EMPNO CHAR(6)  NOT NULL,
          PROJNO CHAR(6)  NOT NULL,
          ACTNO SMALLINT  NOT NULL,
          EMPTIME DEC(5,2),
          EMSTDATE DATE,
          EMENDATE  DATE )
INSERT INTO MA_EMP_ACT
  SELECT * FROM EMP_ACT
    WHERE SUBSTR(PROJNO, 1, 2) = 'MA'
```

Example 5: Use a C program statement to add a skeleton project to the PROJECT table. Obtain the project number (PROJNO), project name (PROJNAME), department number (DEPTNO), and responsible employee (RESPEMP) from host variables. Use the current date as the project start date (PRSTDATE). Assign a NULL value to the remaining columns in the table.

```
EXEC SQL INSERT INTO PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP, PRSTDATE)
  VALUES (:PRJNO, :PRJNM, :DPTNO, :REMP, CURRENT DATE);
```

Example 6: Specify an INSERT statement as the *data-change-table-reference* within a SELECT statement. Define an extra include column whose values are specified in the VALUES clause, which is then used as an ordering column for the inserted rows.

```
SELECT inorder.ordernum
  FROM (INSERT INTO orders(custno)INCLUDE (insertnum integer)
    VALUES(:cnum1, 1), (:cnum2, 2)) InsertedOrders
  ORDER BY insertnum;
```

Example 7: Use a C program statement to add a document to the DOCUMENTS table. Obtain values for the document ID (DOCID) column and the document data (XMLDOC) column from a host variable that binds to an SQL TYPE IS XML AS BLOB_FILE.

```
EXEC SQL INSERT INTO DOCUMENTS
  (DOCID, XMLDOC) VALUES (:docid, :xmldoc)
```

Related reference:

- "CREATE TABLE " on page 956
- "CREATE VIEW " on page 1034
- "Assignments and comparisons" in *SQL Reference, Volume 1*
- "Expressions" in *SQL Reference, Volume 1*
- "SQL queries" on page 1210

Related samples:

- "dtlob.sqc -- How to use the LOB data type (C)"
- "tbident.sqc -- How to use identity columns (C)"
- "tbmod.sqc -- How to modify table data (C)"
- "tbtrig.sqc -- How to use a trigger on a table (C)"
- "dtlob.sqC -- How to use the LOB data type (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "tbtrig.sqC -- How to use a trigger on a table (C++)"
- "DtLob.java -- How to use LOB data type (JDBC)"
- "TbIdent.java -- How to use Identity Columns (JDBC)"
- "TbMod.java -- How to modify table data (JDBC)"
- "TbTrig.java -- How to use triggers (JDBC)"
- "TbIdent.sqlj -- How to use Identity Columns (SQLj)"
- "TbMod.sqlj -- How to modify table data (SQLj)"
- "TbTrig.sqlj -- How to use triggers (SQLj)"
- "updat.sqb -- How to update, delete and insert table data (MF COBOL)"

RENAME

The RENAME statement renames an existing table or index.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- CONTROL privilege on the table or index
- ALTERIN privilege on the schema
- SYSADM or DBADM authority

Syntax:

```
               ┌─TABLE─┐
►►──RENAME──┬──┴───────┴──source-table-name──┬──TO──target-identifier──────────►◄
            └──INDEX──source-index-name──────┘
```

Description:

TABLE *source-table-name*
: Names the existing table that is to be renamed. The name, including the schema name, must identify a table that already exists in the database (SQLSTATE 42704). It must not be the name of a catalog table (SQLSTATE 42832), a materialized query table, a typed table (SQLSTATE 42997), a declared global temporary table (SQLSTATE 42995), a nickname, or an object other than a table or an alias (SQLSTATE 42809). The TABLE keyword is optional.

INDEX *source-index-name*
: Names the existing index that is to be renamed. The name, including the schema name, must identify an index that already exists in the database (SQLSTATE 42704). It must not be the name of an index on a declared global temporary table (SQLSTATE 42995). The schema name must not be SYSIBM, SYSCAT, SYSFUN, or SYSSTAT (SQLSTATE 42832).

target-identifier
: Specifies the new name for the table or index without a schema name. The schema name of the source object is used to qualify the new name for the object. The qualified name must *not* identify a table, view, alias, or index that already exists in the database (SQLSTATE 42710).

Rules:

When renaming a table, the source table must not:
- Be referenced in any existing view definitions or materialized query table definitions
- Be referenced in any triggered SQL statements in existing triggers or be the subject table of an existing trigger
- Be referenced in an SQL function
- Have any check constraints
- Have any generated columns other than the identity column
- Be a parent or dependent table in any referential integrity constraints
- Be the scope of any existing reference column
- Be referenced by an XSR object that has been enabled for decomposition

An error (SQLSTATE 42986) is returned if the source table violates one or more of these conditions.

When renaming an index:

- The source index must not be a system-generated index for an implementation table on which a typed table is based (SQLSTATE 42858).

Notes:

- Catalog entries are updated to reflect the new table or index name.
- *All* authorizations associated with the source table or index name are *transferred* to the new table or index name (the authorization catalog tables are updated appropriately).
- Indexes defined over the source table are *transferred* to the new table (the index catalog tables are updated appropriately).
- RENAME TABLE invalidates any packages that are dependent on the source table. RENAME INDEX invalidates any packages that are dependent on the source index.
- If an alias is used for the *source-table-name*, it must resolve to a table name. The table is renamed within the schema of this table. The alias is not changed by the RENAME statement and continues to refer to the old table name.
- A table with primary key or unique constraints can be renamed if none of the primary key or unique constraints are referenced by any foreign key.

Examples:

Change the name of the EMP table to EMPLOYEE.

```
RENAME TABLE EMP TO EMPLOYEE
RENAME TABLE ABC.EMP TO EMPLOYEE
```

Change the name of the index NEW-IND to IND.

```
RENAME INDEX NEW-IND TO IND
RENAME INDEX ABC.NEW-IND TO IND
```

RENAME TABLESPACE

The RENAME TABLESPACE statement renames an existing table space.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include either SYSCTRL or SYSADM authority.

Syntax:

```
►►──RENAME──TABLESPACE──source-tablespace-name──TO──target-tablespace-name──────►◄
```

Description:

source-tablespace-name
Specifies the existing table space that is to be renamed, as a one-part name. It

is an SQL identifier (either ordinary or delimited). The table space name must identify a table space that already exists in the catalog (SQLSTATE 42704).

target-tablespace-name
Specifies the new name for the table space, as a one-part name. It is an SQL identifier (either ordinary or delimited). The new table space name must *not* identify a table space that already exists in the catalog (SQLSTATE 42710), and it cannot start with 'SYS' (SQLSTATE 42939).

Rules:
- The SYSCATSPACE table space cannot be renamed (SQLSTATE 42832).
- Any table spaces with "rollforward pending" or "rollforward in progress" states cannot be renamed (SQLSTATE 55039)

Notes:
- Renaming a table space will update the minimum recovery time of a table space to the point in time when the rename took place. This implies that a roll forward at the table space level must be to at least this point in time.
- The new table space name must be used when restoring a table space from a backup image, where the rename was done after the backup was created.

Example:

Change the name of the table space USERSPACE1 to DATA2000:

```
RENAME TABLESPACE USERSPACE1 TO DATA2000
```

REVOKE (Database Authorities)

This form of the REVOKE statement revokes authorities that apply to the entire database.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

SYSADM authority is required to revoke any of the following authorities:
- DBADM
- SECADM

Syntax:

```
                    ┌─,───────────────────────────────┐
                    ▼                                 │
►►──REVOKE──────────┬─BINDADD─────────────────────┬───┴──ON DATABASE───────────────────►
                    ├─CONNECT─────────────────────┤
                    ├─CREATETAB───────────────────┤
                    ├─CREATE_EXTERNAL_ROUTINE─────┤
                    ├─CREATE_NOT_FENCED_ROUTINE───┤
                    ├─IMPLICIT_SCHEMA─────────────┤
                    ├─DBADM───────────────────────┤
                    ├─LOAD────────────────────────┤
                    ├─QUIESCE_CONNECT─────────────┤
                    └─SECADM──────────────────────┘

          ┌─,───────────────────────────────────┐
          ▼                                     │   ┌─BY ALL─┐
►─FROM────┬──┬─────────┬──authorization-name──┬─┴───┴────────┴──────────────────►◄
          │  ├─USER────┤                      │
          │  └─GROUP───┘                      │
          └─PUBLIC────────────────────────────┘
```

Description:

BINDADD
: Revokes the authority to create packages. The creator of a package automatically has the CONTROL privilege on that package and retains this privilege even if his BINDADD authority is subsequently revoked.

  The BINDADD authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority.

CONNECT
: Revokes the authority to access the database.

  Revoking the CONNECT authority from a user does not affect any privileges that were granted to that user on objects in the database. If the user is subsequently granted the CONNECT authority again, all previously held privileges are still valid (assuming they were not explicitly revoked).

  The CONNECT authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATETAB
: Revokes the authority to create tables. The creator of a table automatically has the CONTROL privilege on that table, and retains this privilege even if his CREATETAB authority is subsequently revoked.

  The CREATETAB authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

CREATE_EXTERNAL_ROUTINE
: Revokes the authority to register external routines. Once an external routine has been registered, it continues to exist, even if CREATE_EXTERNAL_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

  CREATE_EXTERNAL_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM or CREATE_NOT_FENCED_ROUTINE authority without also revoking DBADM or CREATE_NOT_FENCED_ROUTINE authority (SQLSTATE 42504).

CREATE_NOT_FENCED_ROUTINE
Revokes the authority to register routines that execute in the database manager's process. Once a routine has been registered as not fenced, it continues to run in this manner, even if CREATE_NOT_FENCED_ROUTINE is subsequently revoked from the authorization ID that registered the routine.

CREATE_NOT_FENCED_ROUTINE authority cannot be revoked from an *authorization-name* holding DBADM authority without also revoking the DBADM authority (SQLSTATE 42504).

IMPLICIT_SCHEMA
Revokes the authority to implicitly create a schema. It does not affect the ability to create objects in existing schemas or to process a CREATE SCHEMA statement.

DBADM
Revokes the DBADM authority.

DBADM authority cannot be revoked from PUBLIC (because it cannot be granted to PUBLIC).

CAUTION:
Revoking DBADM authority does not automatically revoke any privileges that were held by the *authorization-name* **on objects in the database, nor does it revoke any of the other database authorities that were implicitly and automatically granted when DBADM authority was originally granted.**

LOAD
Revokes the authority to LOAD in this database.

QUIESCE_CONNECT
Revokes the authority to access the database while it is quiesced.

SECADM
Revokes the security administrator authority.

FROM
Indicates from whom the authorities are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the authorities from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the authorities from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:

- If all rows for the grantee in the SYSCAT.DBAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
- If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
- If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- Revoking a specific privilege does not necessarily revoke the ability to perform an action. A user may proceed with a task if other privileges are held by PUBLIC or a group, or if the user has a higher level authority, such as DBADM.
- ***Compatibilities***

  For compatibility with versions earlier than Version 8, the option CREATE_NOT_FENCED can be substituted for CREATE_NOT_FENCED_ROUTINE.

Examples:

Example 1: Given that USER6 is only a user and not a group, revoke the privilege to create tables from the user USER6.

```
   REVOKE CREATETAB ON DATABASE FROM USER6
```

Example 2: Revoke BINDADD authority on the database from a group named D024. There are two rows in the SYSCAT.DBAUTH catalog view for this grantee; one with a GRANTEETYPE of U and one with a GRANTEETYPE of G.

```
   REVOKE BINDADD ON DATABASE FROM GROUP D024
```

In this case, the GROUP keyword must be specified; otherwise an error will occur (SQLSTATE 56092).

Example 3: Revoke security administrator authority from user Walid.

```
   REVOKE SECADM ON DATABASE FROM USER Walid
```

Related reference:

- "REVOKE (Index Privileges) " on page 1125
- "REVOKE (Package Privileges) " on page 1126
- "REVOKE (Routine Privileges) " on page 1129
- "REVOKE (Schema Privileges) " on page 1132
- "REVOKE (Server Privileges) " on page 1136
- "REVOKE (Table Space Privileges) " on page 1139
- "REVOKE (Table, View, or Nickname Privileges) " on page 1141

Related samples:

- "dbauth.sqc -- How to grant, display, and revoke authorities at database level (C)"
- "dbauth.sqC -- How to grant, display, and revoke authorities at database level (C++)"
- "DbAuth.java -- Grant, display or revoke privileges on database (JDBC)"
- "DbAuth.sqlj -- Grant, display or revoke privileges on database (SQLj)"

REVOKE (Exemption)

This form of the REVOKE statement revokes an exemption to a label-based access control (LBAC) access rule.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
►►──REVOKE EXEMPTION ON RULE──┬─access-rule─┬──FOR──policy-name──────────────────►
                              └─ALL─────────┘

►─FROM USER──authorization-name─────────────────────────────────────────────────►◄
```

Description:

EXEMPTION ON RULE
Revokes the exemption on an access rule.

access-rule
Identifies the rule for which the exemption is to be granted. The rule must be one of the access rules for the security policy.

ALL
Specifies that exemptions are to be revoked for all the access rules for the security policy.

FOR *policy-name*
Specifies the name of the security policy on which exemptions are to be revoked.

FROM USER *authorization-name*
Specifies the authorization ID from which the exemption is being revoked.

Examples:

Example 1: Revoke the exemption on access rule DB2LBACREADSET for security policy DATA_ACCESS from user WALID.

```
REVOKE EXEMPTION ON RULE DB2LBACREADSET FOR DATA_ACCESS
  FROM USER WALID
```

Related concepts:

- "Database authorities" on page 74

Related reference:

- "GRANT (Exemption) " on page 1084

REVOKE (Index Privileges)

This form of the REVOKE statement revokes the CONTROL privilege on an index.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

Syntax:

```
►►──REVOKE CONTROL ON INDEX──index-name─────────────────────────────────►


          ┌─,──────────────────────────────┐
          ▼                                │  ┌─BY ALL─┐
►──FROM──────┬──┬───────┬──authorization-name─┬──┴────────┴──────────────────►◄
             │  ├─USER──┤                     │
             │  └─GROUP─┘                     │
             └─PUBLIC─────────────────────────┘
```

Description:

CONTROL
: Revokes the privilege to drop the index. This is the CONTROL privilege for indexes, which is automatically granted to creators of indexes.

ON INDEX *index-name*
: Specifies the name of the index on which the CONTROL privilege is to be revoked.

FROM
: Indicates from whom the privileges are revoked.

  USER
  : Specifies that the *authorization-name* identifies a user.

  GROUP
  : Specifies that the *authorization-name* identifies a group name.

  authorization-name,...
  : Lists one or more authorization IDs.

    The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

  PUBLIC
  : Revokes the privileges from PUBLIC.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules:
- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.INDEXAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:
- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have authorities such as ALTERIN on the schema of an index.

Examples:

Example 1: Given that USER4 is only a user and not a group, revoke the privilege to drop an index DEPTIDX from the user USER4.

```
REVOKE CONTROL ON INDEX DEPTIDX FROM USER4
```

Example 2: Revoke the privilege to drop an index LUNCHITEMS from the user CHEF and the group WAITERS.

```
REVOKE CONTROL ON INDEX LUNCHITEMS
  FROM USER CHEF, GROUP WAITERS
```

Related reference:
- "REVOKE (Database Authorities) " on page 1120
- "REVOKE (Package Privileges) " on page 1126
- "REVOKE (Routine Privileges) " on page 1129
- "REVOKE (Schema Privileges) " on page 1132
- "REVOKE (Server Privileges) " on page 1136
- "REVOKE (Table Space Privileges) " on page 1139
- "REVOKE (Table, View, or Nickname Privileges) " on page 1141

REVOKE (Package Privileges)

This form of the REVOKE statement revokes CONTROL, BIND, and EXECUTE privileges against a package.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced package
- SYSADM or DBADM authority

DBADM or SYSADM authority is required to revoke the CONTROL privilege.

Syntax:

```
                 ┌─,──────────┐
                 ▼            │
►►──REVOKE────┬─BIND──────┬──┴──────────────────────────────────────────────►
              ├─CONTROL───┤
              │      (1)  │
              └─EXECUTE───┘

               (2)
►──ON──PACKAGE─────┬──────────────┬──package-id──────────────────────────────►
                   └─schema-name.─┘

              ┌─,──────────────────────────────┐
              ▼                                │
►──FROM────┬─┬───────┬──authorization-name─┬─┴──┬────────┬────────────────►◄
           │ ├─USER──┤                    │    └─BY ALL─┘
           │ └─GROUP─┘                    │
           └─PUBLIC───────────────────────┘
```

Notes:

1 RUN can be used as a synonym for EXECUTE.

2 PROGRAM can be used as a synonym for PACKAGE.

Description:

BIND
Revokes the privilege to execute BIND or REBIND on—or to add a new version of— the referenced package.

The BIND privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package, without also revoking the CONTROL privilege.

CONTROL
Revokes the privilege to drop the package and to extend package privileges to other users.

Revoking CONTROL does not revoke the other package privileges.

EXECUTE
Revokes the privilege to execute the package.

The EXECUTE privilege cannot be revoked from an *authorization-name* that holds CONTROL privilege on the package without also revoking the CONTROL privilege.

ON PACKAGE *schema-name.package-id*
Specifies the name of the package on which privileges are to be revoked. If a

schema name is not specified, the package ID is implicitly qualified by the default schema. The revoking of a package privilege applies to all versions of the package.

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.PACKAGEAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have privileges such as ALTERIN on the schema of a package.

Examples:

Example 1: Revoke the EXECUTE privilege on package CORPDATA.PKGA from PUBLIC.

```
REVOKE EXECUTE
  ON PACKAGE CORPDATA.PKGA
  FROM PUBLIC
```

Example 2: Revoke CONTROL authority on the RRSP_PKG package for the user FRANK and for PUBLIC.

```
REVOKE CONTROL
  ON PACKAGE RRSP_PKG
  FROM USER FRANK, PUBLIC
```

Related reference:

- “REVOKE (Database Authorities) ” on page 1120
- “REVOKE (Index Privileges) ” on page 1125
- “REVOKE (Routine Privileges) ” on page 1129
- “REVOKE (Schema Privileges) ” on page 1132
- “REVOKE (Server Privileges) ” on page 1136
- “REVOKE (Table Space Privileges) ” on page 1139
- “REVOKE (Table, View, or Nickname Privileges) ” on page 1141

REVOKE (Routine Privileges)

This form of the REVOKE statement revokes privileges on a routine (function, method, or procedure).

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

Syntax:

```
►►──REVOKE EXECUTE ON──┬─┤ function-designator ├───────┬──────────────────►
                       ├─FUNCTION──┬─────────┬──*──────┤
                       │           └─schema.─┘         │
                       ├─┤ method-designator ├─────────┤
                       ├─METHOD * FOR──┬─type-name───┬─┤
                       │               └─┬───────┬─*─┘ │
                       │                 └schema.┘     │
                       ├─┤ procedure-designator ├──────┤
                       └─PROCEDURE──┬─────────┬──*─────┘
                                    └─schema.─┘


           ┌─,──────────────────────────────────┐
           ▼                                    │  ┌─BY ALL─┐
►─FROM─────┬──┬───────┬──authorization-name─┬──┴──┴────────┴──RESTRICT──────►◄
           │  ├─USER──┤                     │
           │  └─GROUP─┘                     │
           └─PUBLIC─────────────────────────┘
```

Description:

EXECUTE
: Revokes the privilege to run the identified user-defined function, method, or procedure.

function-designator
: Uniquely identifies the function.

FUNCTION *schema.**
Identifies the explicit grant for all the existing and future functions in the schema. Revoking the *schema*.* privilege does not revoke any privileges that were granted on a specific function. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

method-designator
Uniquely identifies the method.

METHOD *
Identifies the explicit grant for all the existing and future methods for the type *type-name*. Revoking the * privilege does not revoke any privileges that were granted on a specific method.

FOR *type-name*
Names the type in which the specified method is found. The name must identify a type already described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the value of the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified type names. An asterisk (*) can be used in place of *type-name* to identify the explicit grant on all existing and future methods for all existing and future types in the schema. Revoking the privilege using an asterisk for method and *type-name* does not revoke any privileges that were granted on a specific method or on all methods for a specific type.

procedure-designator
Uniquely identifies the procedure.

PROCEDURE *schema.**
Identifies the explicit grant for all the existing and future procedures in the schema. Revoking the *schema*.* privilege does not revoke any privileges that were granted on a specific procedure. In dynamic SQL statements, if a schema is not specified, the schema in the CURRENT SCHEMA special register will be used. In static SQL statements, if a schema is not specified, the schema in the QUALIFIER precompile/bind option will be used.

FROM
Specifies from whom the EXECUTE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists the authorization IDs of one or more users or groups. The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

PUBLIC
Revokes the EXECUTE privilege from all users.

BY ALL
Revokes the EXECUTE privilege from all named users who were explicitly granted the privilege, regardless of who granted it. This is the default behavior.

RESTRICT
Specifies that the EXECUTE privilege cannot be revoked if both of the following are true (SQLSTATE 42893):
- The specified routine is used in a view, trigger, constraint, index extension, SQL function, SQL method, transform group, or is referenced as the SOURCE of a sourced function.
- The loss of the EXECUTE privilege would cause the definer of the view, trigger, constraint, index extension, SQL function, SQL method, transform group, or sourced function to no longer be able to execute the specified routine.

Rules:
- It is not possible to revoke the EXECUTE privilege on a function or method defined with schema ’SYSIBM’ or ’SYSFUN’ (SQLSTATE 42832).
- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.ROUTINEAUTH catalog views have a GRANTEETYPE of U, then USER is assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP is assumed.
  - If some rows have U and some rows have G, an error (SQLSTATE 56092) is raised.

Examples:

Example 1: Revoke the EXECUTE privilege on function CALC_SALARY from user JONES. Assume that there is only one function in the schema with function name CALC_SALARY.

```
REVOKE EXECUTE ON FUNCTION CALC_SALARY FROM JONES RESTRICT
```

Example 2: Revoke the EXECUTE privilege on procedure VACATION_ACCR from all users at the current server.

```
REVOKE EXECUTE ON PROCEDURE VACATION_ACCR FROM PUBLIC RESTRICT
```

Example 3: Revoke the EXECUTE privilege on function NEW_DEPT_HIRES from HR (Human Resources). The function has two input parameters of type INTEGER and CHAR(10), respectively. Assume that the schema has more than one function named NEW_DEPT_HIRES.

```
REVOKE EXECUTE ON FUNCTION NEW_DEPT_HIRES (INTEGER, CHAR(10))
  FROM HR RESTRICT
```

Example 4: Revoke the EXECUTE privilege on method SET_SALARY for type EMPLOYEE from user Jones.

```
REVOKE EXECUTE ON METHOD SET_SALARY FOR EMPLOYEE FROM JONES RESTRICT
```

Related reference:
- “Function, method, and procedure designators” on page 1286
- “REVOKE (Database Authorities) ” on page 1120
- “REVOKE (Index Privileges) ” on page 1125
- “REVOKE (Package Privileges) ” on page 1126
- “REVOKE (Schema Privileges) ” on page 1132
- “REVOKE (Server Privileges) ” on page 1136
- “REVOKE (Table Space Privileges) ” on page 1139
- “REVOKE (Table, View, or Nickname Privileges) ” on page 1141

REVOKE (Schema Privileges)

This form of the REVOKE statement revokes the privileges on a schema.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

Syntax:

```
                 ┌─,──────────┐
                 ▼            │
►►──REVOKE────────┬─ALTERIN──┬┴──ON SCHEMA──schema-name──────────────────────────►
                  ├─CREATEIN─┤
                  └─DROPIN───┘

            ┌─,───────────────────────────┐
            ▼                             │  ┌─BY ALL─┐
►──FROM─────┬──┬───────┬──authorization-name─┬┴──┴────────┴─────────────────────────►◄
            │  ├─USER──┤                    │
            │  └─GROUP─┘                    │
            └─PUBLIC────────────────────────┘
```

Description:

ALTERIN
: Revokes the privilege to alter or comment on objects in the schema.

CREATEIN
: Revokes the privilege to create objects in the schema.

DROPIN
: Revokes the privilege to drop objects in the schema.

ON SCHEMA *schema-name*
: Specifies the name of the schema on which privileges are to be revoked.

FROM
: Indicates from whom the privileges are revoked.

  USER
  : Specifies that the *authorization-name* identifies a user.

  GROUP
  : Specifies that the *authorization-name* identifies a group name.

  authorization-name,...
  : Lists one or more authorization IDs.

    The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.SCHEMAAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have a higher level authority such as DBADM.

Examples:

Example 1: Given that USER4 is only a user and not a group, revoke the privilege to create objects in schema DEPTIDX from the user USER4.

```
REVOKE CREATEIN ON SCHEMA DEPTIDX FROM USER4
```

Example 2: Revoke the privilege to drop objects in schema LUNCH from the user CHEF and the group WAITERS.

```
REVOKE DROPIN ON SCHEMA LUNCH
  FROM USER CHEF, GROUP WAITERS
```

Related reference:

- “REVOKE (Database Authorities) ” on page 1120
- “REVOKE (Index Privileges) ” on page 1125
- “REVOKE (Package Privileges) ” on page 1126
- “REVOKE (Routine Privileges) ” on page 1129
- “REVOKE (Server Privileges) ” on page 1136
- “REVOKE (Table Space Privileges) ” on page 1139
- “REVOKE (Table, View, or Nickname Privileges) ” on page 1141

REVOKE (Security Label)

This form of the REVOKE statement revokes a label-based access control (LBAC) security label.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
►►──REVOKE SECURITY LABEL──security-label-name──FROM USER──authorization-name────────►◄
```

Description:

SECURITY LABEL *security-label-name*
: Revokes the security label *security-label-name*. The *security-label-name* must be qualified with a security policy and must identify a security label that exists at the current server (SQLSTATE 42704), and that is held by *authorization-name* (SQLSTATE 42504).

FROM USER *authorization-name*
: Specifies the authorization ID from which the security label is being revoked.

Examples:

Example 1: Revoke the security label EMPLOYEESECLABEL, which is part of the security policy DATA_ACCESS, from user WALID.

```
   REVOKE SECURITY LABEL DATA_ACCESS.EMPLOYEESECLABEL
     FROM USER WALID
```

Related concepts:

- "Database authorities" on page 74

Related reference:

- "GRANT (Security Label) " on page 1094

REVOKE (Sequence Privileges)

This form of the REVOKE statement revokes privileges on a sequence.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared. However, if the bind option DYNAMICRULES BIND applies, the statement cannot be dynamically prepared (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

Syntax:

```
                ┌─,──────────┐
                ▼            │
►►──REVOKE────┬─ALTER─┬──────┴──ON SEQUENCE──sequence-name──────────────────►
              └─USAGE─┘
```

```
                ┌─,──────────────────────────────┐
                ▼                                 │  ┌─RESTRICT─┐
►─FROM──────────┬────┬─────────┬─authorization-name─┴──┴──────────┴──────────────►◄
                │    ├─USER──┤                     │
                │    └─GROUP─┘                     │
                └─PUBLIC────────────────────────────┘
```

Description:

ALTER
Revokes the privilege to change the properties of a sequence or to restart sequence number generation using the ALTER SEQUENCE statement.

USAGE
Revokes the privilege to reference a sequence using *nextval-expression* or *prevval-expression*.

ON SEQUENCE *sequence-name*
Identifies the sequence on which the specified privileges are to be revoked. The sequence name, including an implicit or explicit schema qualifier, must uniquely identify an existing sequence at the current server. If no sequence by this name exists, an error is returned (SQLSTATE 42704).

FROM
Specifies from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group.

authorization-name,...
Lists the authorization IDs of one or more users or groups. The authorization ID of the REVOKE statement itself cannot be specified (SQLSTATE 42502).

PUBLIC
Revokes the specified privileges from all users.

RESTRICT
This optional keyword indicates that the statement will fail if any objects depend on the privilege being revoked.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.SEQUENCEAUTH catalog view have a GRANTEETYPE of U, then USER is assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP is assumed.
  - If some rows have U and some rows have G, an error is returned (SQLSTATE 56092).

Notes:

- Revoking a specific privilege does not necessarily remove the ability to perform an action. A user can proceed if other privileges are held by PUBLIC or by a group to which the user belongs, or if the user has a higher level of authority, such as DBADM.

Examples:

Example 1: Revoke the USAGE privilege on a sequence called GENERATE_ID from user ENGLES. There is one row in the SYSCAT.SEQUENCEAUTH catalog view for this sequence and grantee, and the GRANTEETYPE value is U.

```
REVOKE USAGE ON SEQUENCE GENERATE_ID FROM ENGLES
```

Example 2: Revoke alter privileges on sequence GENERATE_ID that were previously granted to all local users. (Grants to specific users are not affected.)

```
REVOKE ALTER ON SEQUENCE GENERATE_ID FROM PUBLIC
```

Example 3: Revoke all privileges on sequence GENERATE_ID from users PELLOW and MLI, and from group PLANNERS.

```
REVOKE ALTER, USAGE ON SEQUENCE GENERATE_ID
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

Related reference:

- "GRANT (Sequence Privileges) " on page 1096
- "REVOKE (Database Authorities) " on page 1120
- "REVOKE (Index Privileges) " on page 1125
- "REVOKE (Package Privileges) " on page 1126
- "REVOKE (Routine Privileges) " on page 1129
- "REVOKE (Schema Privileges) " on page 1132
- "REVOKE (Server Privileges) " on page 1136
- "REVOKE (Table Space Privileges) " on page 1139
- "REVOKE (Table, View, or Nickname Privileges) " on page 1141

REVOKE (Server Privileges)

This form of the REVOKE statement revokes the privilege to access and use a specified data source in pass-through mode.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM or SYSADM authority.

Syntax:

```
►►──REVOKE PASSTHRU ON SERVER──server-name──FROM─────────────────────────────►
```

```
     ┌─,──────────────────────────────────┐
     ▼                                    │  ┌─BY ALL─┐
►─────┬─────────┬──authorization-name──┬──┴──┴────────┴──────────────────────►◄
      │ ├─USER──┤                      │
      │ └─GROUP─┘                      │
      └─PUBLIC─────────────────────────┘
```

Description:

SERVER *server-name*
: Names the data source for which the privilege to use in pass-through mode is being revoked. *server-name* must identify a data source that is described in the catalog.

FROM
: Specifies from whom the privilege is revoked.

  USER
  : Specifies that the *authorization-name* identifies a user.

  GROUP
  : Specifies that the *authorization-name* identifies a group name.

  authorization-name,...
  : Lists the authorization IDs of one or more users or groups.

    The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

  PUBLIC
  : Revokes from all users the privilege to pass through to *server-name*.

BY ALL
: Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Examples:

Example 1: Revoke USER6's privilege to pass through to data source MOUNTAIN.

```
REVOKE PASSTHRU ON SERVER MOUNTAIN FROM USER USER6
```

Example 2: Revoke group D024's privilege to pass through to data source EASTWING.

```
REVOKE PASSTHRU ON SERVER EASTWING FROM GROUP D024
```

The members of group D024 will no longer be able to use their group ID to pass through to EASTWING. But if any members have the privilege to pass through to EASTWING under their own user IDs, they will retain this privilege.

Related reference:

- "REVOKE (Database Authorities) " on page 1120
- "REVOKE (Index Privileges) " on page 1125
- "REVOKE (Package Privileges) " on page 1126
- "REVOKE (Routine Privileges) " on page 1129
- "REVOKE (Schema Privileges) " on page 1132
- "REVOKE (Table Space Privileges) " on page 1139

- "REVOKE (Table, View, or Nickname Privileges) " on page 1141

REVOKE (SETSESSIONUSER Privilege)

This form of the REVOKE statement revokes one or more SETSESSIONUSER privileges from one or more authorization IDs.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include SECADM authority.

Syntax:

```
                              ┌─,─────────────────────────────────┐        ┌─,─────────────────────────────────┐
                              ▼                                   │        ▼                                   │
►►──REVOKE SETSESSIONUSER ON──────┬─USER──session-authorization-name─┬─┴──FROM────┬─USER──┬──authorization-name─┴──────────►◄
                                  └─PUBLIC───────────────────────────┘             └─GROUP─┘
```

Description:

SETSESSIONUSER ON
: Revokes the privilege to assume the identity of a new authorization ID.

USER *session-authorization-name*
: Specifies the authorization ID that the *authorization-name* is able to assume, using the SET SESSION AUTHORIZATION statement. The *session-authorization-name* must identify a user that the *authorization-name* can assume, not a group (SQLSTATE 42504).

PUBLIC
: Specifies that all privileges to set the session authorization will be revoked.

FROM
: Specifies from whom the privilege is revoked.

  USER
  : Specifies that the *authorization-name* identifies a user.

  GROUP
  : Specifies that the *authorization-name* identifies a group.

  authorization-name,...
  : Lists the authorization IDs of one or more users or groups.

    The list of authorization IDs cannot include the authorization ID of the user issuing the statement (SQLSTATE 42502).

Examples:

Example 1: User PAUL holds the privilege to set the session authorization to WALID and therefore to execute SQL statements as user WALID. The following statement revokes that privilege.

```
REVOKE SETSESSIONUSER ON USER WALID
  FROM USER PAUL
```

Example 2: User GUYLAINE holds the privilege to set the session authorization to BOBBY, RICK, or KEVIN and therefore to execute SQL statements as BOBBY, RICK, or KEVIN. The following statement revokes the privilege to use two of those authorization IDs. After this statement executes, GUYLAINE will only be able to set the session authorization to KEVIN.

```
REVOKE SETSESSIONUSER ON USER BOBBY, USER RICK
  FROM USER GUYLAINE
```

Example 3: The group ACCTG and user WALID can set session authorization to any authorization ID. The following statement revokes that privilege from both ACCTG and WALID.

```
REVOKE SETSESSIONUSER ON PUBLIC
  FROM USER WALID, GROUP ACCTG
```

Related concepts:
- “Database authorities” on page 74

Related reference:
- “GRANT (SETSESSIONUSER Privilege) ” on page 1100

REVOKE (Table Space Privileges)

This form of the REVOKE statement revokes the USE privilege on a table space.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include DBADM, SYSCTRL, or SYSADM authority.

Syntax:

```
>>--REVOKE USE OF TABLESPACE--tablespace-name--FROM------------------------->


    .-,-----------------------------------.
    V                                     |   .-BY ALL-.
>------+-----------+--authorization-name-+-+---+--------+------------------><
     | +-USER--+ |                       |
     | '-GROUP-' |                       |
     '-PUBLIC----------------------------'
```

Description:

USE
Revokes the privilege to specify or default to the table space when creating a table.

OF TABLESPACE *tablespace-name*
Specifies the table space on which the USE privilege is to be revoked. The table space cannot be SYSCATSPACE (SQLSTATE 42838) or a SYSTEM TEMPORARY table space (SQLSTATE 42809).

FROM
Indicates from whom the USE privilege is revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name
Lists one or more authorization IDs.

The authorization ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the USE privilege from PUBLIC.

BY ALL
Revokes the privilege from all named users who were explicitly granted that privilege, regardless of who granted it. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.TBSPACEAUTH catalog view have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error results (SQLSTATE 56092).

Notes:

- Revoking the USE privilege does not necessarily revoke the ability to create tables in that table space. A user may still be able to create tables in that table space if the USE privilege is held by PUBLIC or a group, or if the user has a higher level authority, such as DBADM.

Examples:

Example 1: Revoke the privilege to create tables in table space PLANS from the user BOBBY.

```
REVOKE USE OF TABLESPACE PLANS FROM USER BOBBY
```

Related reference:

- "REVOKE (Database Authorities) " on page 1120
- "REVOKE (Index Privileges) " on page 1125
- "REVOKE (Package Privileges) " on page 1126

- “REVOKE (Routine Privileges) ” on page 1129
- “REVOKE (Schema Privileges) ” on page 1132
- “REVOKE (Server Privileges) ” on page 1136
- “REVOKE (Table, View, or Nickname Privileges) ” on page 1141

REVOKE (Table, View, or Nickname Privileges)

This form of the REVOKE statement revokes privileges on a table, view, or nickname.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

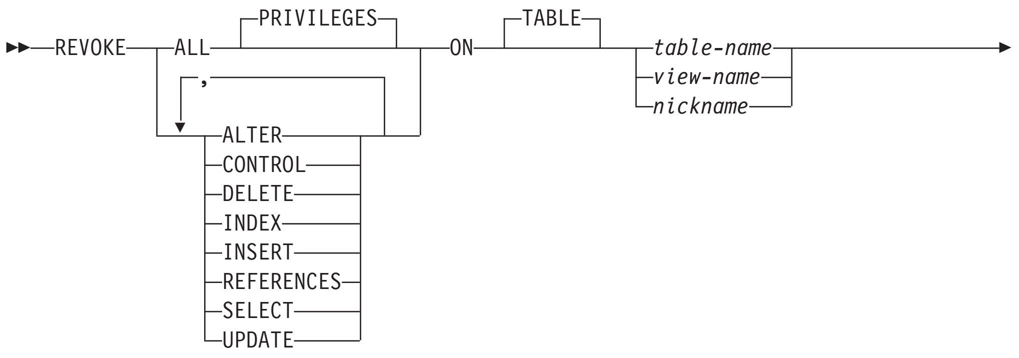
Authorization:

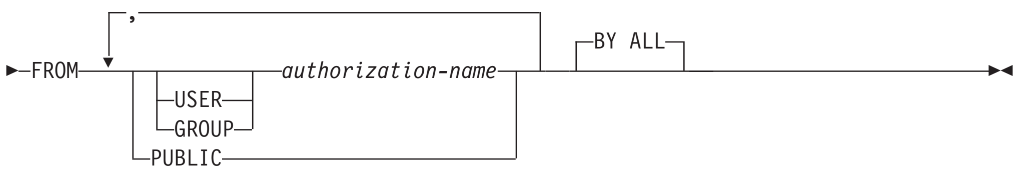
The privileges held by the authorization ID of the statement must include at least one of the following:

- CONTROL privilege on the referenced table, view, or nickname
- SYSADM or DBADM authority

DBADM or SYSADM authority is required to revoke the CONTROL privilege, or to revoke privileges on catalog tables and views.

Syntax:





Description:

ALL or **ALL PRIVILEGES**
: Revokes all privileges (except CONTROL) held by an authorization-name for the specified tables, views, or nicknames.

If ALL is not used, one or more of the keywords listed below must be used. Each keyword revokes the privilege described, but only as it applies to the tables, views, or nicknames named in the ON clause. The same keyword must not be specified more than once.

ALTER
Revokes the privilege to add columns to the base table definition; create or drop a primary key or unique constraint on the table; create or drop a foreign key on the table; add/change a comment on the table, view, or nickname; create or drop a check constraint; create a trigger; add, reset, or drop a column option for a nickname; or, change nickname column names or data types.

CONTROL
Revokes the ability to drop the table, view, or nickname, and the ability to execute the RUNSTATS utility on the table and indexes.

Revoking CONTROL privilege from an *authorization-name* does not revoke other privileges granted to the user on that object.

DELETE
Revokes the privilege to delete rows from the table, updatable view, or nickname.

INDEX
Revokes the privilege to create an index on the table or an index specification on the nickname. The creator of an index or index specification automatically has the CONTROL privilege over the index or index specification (authorizing the creator to drop the index or index specification). In addition, the creator retains this privilege even if the INDEX privilege is revoked.

INSERT
Revokes the privileges to insert rows into the table, updatable view, or nickname, and to run the IMPORT utility.

REFERENCES
Revokes the privilege to create or drop a foreign key referencing the table as the parent. Any column level REFERENCES privileges are also revoked.

SELECT
Revokes the privilege to retrieve rows from the table or view, to create a view on a table, and to run the EXPORT utility against the table or view.

Revoking SELECT privilege may cause some views to be marked inoperative. (For information on inoperative views, see “CREATE VIEW”.)

UPDATE
Revokes the privilege to update rows in the table, updatable view, or nickname. Any column level UPDATE privileges are also revoked.

ON TABLE *table-name* or *view-name* or *nickname*
Specifies the table, view, or nickname on which privileges are to be revoked. The *table-name* cannot be a declared temporary table (SQLSTATE 42995).

FROM
Indicates from whom the privileges are revoked.

USER
Specifies that the *authorization-name* identifies a user.

GROUP
Specifies that the *authorization-name* identifies a group name.

authorization-name,...
Lists one or more authorization IDs.

The ID of the REVOKE statement itself cannot be used (SQLSTATE 42502). It is not possible to revoke the privileges from an *authorization-name* that is the same as the authorization ID of the REVOKE statement.

PUBLIC
Revokes the privileges from PUBLIC.

BY ALL
Revokes each named privilege from all named users who were explicitly granted those privileges, regardless of who granted them. This is the default behavior.

Rules:

- If neither USER nor GROUP is specified, then:
  - If all rows for the grantee in the SYSCAT.TABAUTH and SYSCAT.COLAUTH catalog views have a GRANTEETYPE of U, then USER will be assumed.
  - If all rows have a GRANTEETYPE of G, then GROUP will be assumed.
  - If some rows have U and some rows have G, then an error (SQLSTATE 56092) is raised.

Notes:

- If a privilege is revoked from the *authorization-name* used to create a view (this is called the view's DEFINER in SYSCAT.VIEWS), that privilege is also revoked from any dependent views.
- If the DEFINER of the view loses a SELECT privilege on some object on which the view definition depends (or an object upon which the view definition depends is dropped, or made inoperative in the case of another view), the view will be made inoperative.

  However, if a DBADM or SYSADM explicitly revokes all privileges on the view from the DEFINER, then the record of the DEFINER will not appear in SYSCAT.TABAUTH but nothing will happen to the view - it remains operative.
- Privileges on inoperative views cannot be revoked.
- All packages dependent upon an object for which a privilege is revoked are marked invalid. A package remains invalid until a bind or rebind operation on the application is successfully executed, or the application is executed and the database manager successfully rebinds the application (using information stored in the catalogs). Packages marked invalid due to a revoke may be successfully rebound without any additional grants.

  For example, if a package owned by USER1 contains a SELECT from table T1 and the SELECT privilege for table T1 is revoked from USER1, then the package will be marked invalid. If SELECT authority is re-granted, or if the user holds DBADM authority, the package is successfully rebound when executed.
- Packages, triggers or views that include the use of OUTER(*Z*) in the FROM clause, are dependent on having SELECT privilege on every subtable or subview of *Z*. Similarly, packages, triggers, or views that include the use of DEREF(*Y*) where *Y* is a reference type with a target table or view *Z*, are dependent on having SELECT privilege on every subtable or subview of *Z*. If one of these SELECT privileges is revoked, such packages are invalidated and such triggers or views are made inoperative.

- Table, view, or nickname privileges cannot be revoked from an *authorization-name* with CONTROL on the object without also revoking the CONTROL privilege (SQLSTATE 42504).
- Revoking a specific privilege does not necessarily revoke the ability to perform the action. A user may proceed with their task if other privileges are held by PUBLIC or a group, or if they have privileges such as ALTERIN on the schema of a table or a view.
- If the DEFINER of the materialized query table loses a SELECT privilege on a table on which the materialized query table definition depends (or a table upon which the materialized query table definition depends is dropped), the materialized query table will be dropped.

  However, if a DBADM or SYSADM explicitly revokes all privileges on the materialized query table from the DEFINER, then the record in SYSTABAUTH for the DEFINER will be deleted, but nothing will happen to the materialized query table - it remains operative.
- Revoking nickname privileges has no affect on data source object (table or view) privileges.
- Revoking the SELECT privilege for a table or view that is directly or indirectly referenced in an SQL function or method body may fail if the SQL function or method body cannot be dropped because some other object is dependent on it (SQLSTATE 42893).
- If the DEFINER of the SQL function or method body loses the SELECT privilege on some object on which the function or method body definition depends (or if an object upon which the function or method body definition depends is dropped), the function or method body will be dropped, unless another object depends on the function or method (SQLSTATE 42893).

Examples:

Example 1: Revoke SELECT privilege on table EMPLOYEE from user ENGLES. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
  ON TABLE EMPLOYEE
  FROM ENGLES
```

Example 2: Revoke update privileges on table EMPLOYEE previously granted to all local users. Note that grants to specific users are not affected.

```
REVOKE UPDATE
  ON EMPLOYEE
  FROM PUBLIC
```

Example 3: Revoke all privileges on table EMPLOYEE from users PELLOW and MLI and from group PLANNERS.

```
REVOKE ALL
  ON EMPLOYEE
  FROM USER PELLOW, USER MLI, GROUP PLANNERS
```

Example 4: Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a user named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is U.

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

or

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM USER JOHN
```

Note that an attempt to revoke the privilege from GROUP JOHN would result in an error, since the privilege was not previously granted to GROUP JOHN.

Example 5: Revoke SELECT privilege on table CORPDATA.EMPLOYEE from a group named JOHN. There is one row in the SYSCAT.TABAUTH catalog view for this table and grantee and the GRANTEETYPE value is G.

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM JOHN
```

or

```
REVOKE SELECT
  ON CORPDATA.EMPLOYEE FROM GROUP JOHN
```

Example 6: Revoke user SHAWN's privilege to create an index specification on nickname ORAREM1.

```
REVOKE INDEX
  ON ORAREM1 FROM USER SHAWN
```

Related reference:

- "CREATE TABLE " on page 956
- "CREATE VIEW " on page 1034
- "DROP " on page 1054
- "REVOKE (Database Authorities) " on page 1120
- "REVOKE (Index Privileges) " on page 1125
- "REVOKE (Package Privileges) " on page 1126
- "REVOKE (Routine Privileges) " on page 1129
- "REVOKE (Schema Privileges) " on page 1132
- "REVOKE (Server Privileges) " on page 1136
- "REVOKE (Table Space Privileges) " on page 1139

Related samples:

- "tbpriv.sqc -- How to grant, display, and revoke privileges (C)"
- "tbpriv.sqC -- How to grant, display, and revoke privileges (C++)"
- "TbPriv.java -- How to grant, display and revoke privileges on a table (JDBC)"
- "TbPriv.sqlj -- How to grant, display and revoke privileges on a table (SQLj)"

UPDATE

The UPDATE statement updates the values of specified columns in rows of a table, view or nickname, or the underlying tables, nicknames, or views of the specified fullselect. Updating a row of a view updates a row of its base table, if no INSTEAD OF trigger is defined for the update operation on this view. If such a trigger is defined, the trigger will be executed instead. Updating a row using a nickname updates a row in the data source object to which the nickname refers.

The forms of this statement are:

- The *Searched* UPDATE form is used to update one or more rows (optionally determined by a search condition).

- The *Positioned* UPDATE form is used to update exactly one row (as determined by the current position of a cursor).

Invocation:

An UPDATE statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- UPDATE privilege on the target table, view, or nickname
- UPDATE privilege on each of the columns that are to be updated
- CONTROL privilege on the target table, view, or nickname
- SYSADM or DBADM authority

If a *row-fullselect* is included in the assignment, the privileges held by the authorization ID of the statement must include at least one of the following for each referenced table, view, or nickname:
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority

For each table, view, or nickname referenced by a subquery, the privileges held by the authorization ID of the statement must also include at least one of the following:
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority

If the package used to process the statement is precompiled with SQL92 rules (option LANGLEVEL with a value of SQL92E or MIA), and the searched form of an UPDATE statement includes a reference to a column of the table, view, or nickname in the right hand side of the *assignment-clause*, or anywhere in the *search-condition*, the privileges held by the authorization ID of the statement must also include at least one of the following:
- SELECT privilege
- CONTROL privilege
- SYSADM or DBADM authority

If the specified table or view is preceded by the ONLY keyword, the privileges held by the authorization ID of the statement must also include the SELECT privilege for every subtable or subview of the specified table or view.

GROUP privileges are not checked for static UPDATE statements.

If the target of the update operation is a nickname, privileges on the object at the data source are not considered until the statement is executed at the data source. At this time, the authorization ID that is used to connect to the data source must have the privileges that are required for the operation on the object at the data source. The authorization ID of the statement can be mapped to a different authorization ID at the data source.

Syntax:

searched-update:

```
►►──UPDATE──┬─table-name──────────────┬──┬──────────────────────┬──────────►
            ├─view-name───────────────┤  └─┤ correlation-clause ├─┘
            ├─nickname────────────────┤
            ├─ONLY──(──┬─table-name─┬──)─┤
            │          └─view-name──┘    │
            └─(──fullselect──)────────┘

►──┬───────────────────┬──SET──┤ assignment-clause ├────────────────────────►
   └─┤ include-columns ├─┘

►──┬─────────────────────────┬──┬───────────────┬──────────────────────────►◄
   └─WHERE──search-condition─┘  └─WITH──┬─RR─┬─┘
                                        ├─RS─┤
                                        ├─CS─┤
                                        └─UR─┘
```

positioned-update:

```
►►──UPDATE──┬─table-name──────────────┬──┬──────────────────────┬──────────►
            ├─view-name───────────────┤  └─┤ correlation-clause ├─┘
            ├─nickname────────────────┤
            └─ONLY──(──┬─table-name─┬──)─┘
                       └─view-name──┘

►──SET──┤ assignment-clause ├──WHERE CURRENT OF──cursor-name──────────────►◄
```

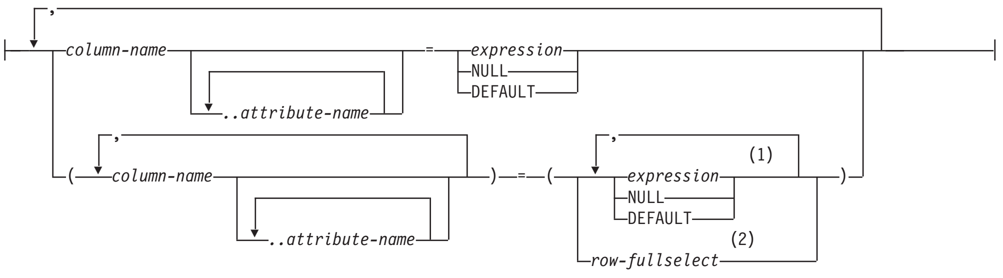
correlation-clause:

```
   ┌─AS─┐
├──┴────┴──correlation-name──┬──────────────────────────┬─────────────────┤
                             │    ┌─,───────────┐        │
                             └─(──▼─column-name─┴──)─────┘
```

include-columns:

```
                  ┌─,─────────────────────────┐
├──INCLUDE──(──▼─column-name──data-type─┴──)─────────────────────────────┤
```

assignment-clause:



Notes:

1 The number of expressions, NULLs and DEFAULTs must match the number of column names.

2 The number of columns in the select list must match the number of column names.

Description:

*table-name***,** *view-name***,** *nickname***, or** *(fullselect)*
: Identifies the object of the update operation. The name must identify a table, view, or nickname described in the catalog, but not a catalog table, a view of a catalog table (unless it is one of the updatable SYSSTAT views), a system-maintained materialized query table, or a read-only view that has no INSTEAD OF trigger defined for its update operations.

If *table-name* is a typed table, rows of the table or any of its proper subtables may get updated by the statement. Only the columns of the specified table may be set or referenced in the WHERE clause. For a positioned UPDATE, the associated cursor must also have specified the same table, view or nickname in the FROM clause without using ONLY.

If the object of the update operation is a fullselect, the fullselect must be updatable, as defined in the "Updatable views" Notes item in the description of the CREATE VIEW statement.

ONLY (*table-name***)**
: Applicable to typed tables, the ONLY keyword specifies that the statement should apply only to data of the specified table and rows of proper subtables cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the table in the FROM clause using ONLY. If *table-name* is not a typed table, the ONLY keyword has no effect on the statement.

ONLY (*view-name***)**
: Applicable to typed views, the ONLY keyword specifies that the statement should apply only to data of the specified view and rows of proper subviews cannot be updated by the statement. For a positioned UPDATE, the associated cursor must also have specified the view in the FROM clause using ONLY. If *view-name* is not a typed view, the ONLY keyword has no effect on the statement.

correlation-clause
: Can be used within *search-condition* or *assignment-clause* to designate a table, view, nickname, or fullselect. For a description of *correlation-clause*, see "table-reference" in the description of "Subselect".

include-columns
: Specifies a set of columns that are included, along with the columns of *table-name* or *view-name*, in the intermediate result table of the UPDATE

statement when it is nested in the FROM clause of a fullselect. The *include-columns* are appended at the end of the list of columns that are specified for *table-name* or *view-name*.

INCLUDE
Specifies a list of columns to be included in the intermediate result table of the UPDATE statement.

column-name
Specifies a column of the intermediate result table of the UPDATE statement. The name cannot be the same as the name of another include column or a column in *table-name* or *view-name* (SQLSTATE 42711).

data-type
Specifies the data type of the include column. The data type must be one that is supported by the CREATE TABLE statement.

SET
Introduces the assignment of values to column names.

assignment-clause

column-name
Identifies a column to be updated. The *column-name* must identify an updatable column of the specified table, view, or nickname, or identify an INCLUDE column. The object ID column of a typed table is not updatable (SQLSTATE 428DZ). A column must not be specified more than once, unless it is followed by *..attribute-name* (SQLSTATE 42701).

If it specifies an INCLUDE column, the column name cannot be qualified.

For a Positioned UPDATE:

- If the *update-clause* was specified in the *select-statement* of the cursor, each column name in the *assignment-clause* must also appear in the *update-clause*.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL MIA or SQL92E was specified when the application was precompiled, the name of any updatable column may be specified.
- If the *update-clause* was not specified in the *select-statement* of the cursor and LANGLEVEL SAA1 was specified either explicitly or by default when the application was precompiled, no columns may be updated.

..attribute-name
Specifies the attribute of a structured type that is set (referred to as an *attribute assignment*. The *column-name* specified must be defined with a user-defined structured type (SQLSTATE 428DP). The attribute-name must be an attribute of the structured type of *column-name* (SQLSTATE 42703). An assignment that does not involve the *..attribute-name* clause is referred to as a *conventional assignment*.

expression
Indicates the new value of the column. The expression is any expression of the type described in “Expressions”. The expression cannot include a column function except when it occurs within a scalar fullselect (SQLSTATE 42903).

An *expression* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated.

An expression cannot contain references to an INCLUDE column.

NULL
Specifies the null value and can only be specified for nullable columns (SQLSTATE 23502). NULL cannot be the value in an attribute assignment (SQLSTATE 429B9) unless it is specifically cast to the data type of the attribute.

DEFAULT
Specifies that the default value should be used based on how the corresponding column is defined in the table. The value that is inserted depends on how the column was defined.

- If the column was defined as a generated column based on an expression, the column value will be generated by the system, based on the expression.
- If the column was defined using the IDENTITY clause, the value is generated by the database manager.
- If the column was defined using the WITH DEFAULT clause, the value is set to the default defined for the column (see *default-clause* in "ALTER TABLE").
- If the column was defined using the NOT NULL clause and the GENERATED clause was not used, or the WITH DEFAULT clause was not used, or DEFAULT NULL was used, the DEFAULT keyword cannot be specified for that column (SQLSTATE 23502).

The only value that a generated column defined with the GENERATED ALWAYS clause can be set to is DEFAULT (SQLSTATE 428C9).

The DEFAULT keyword cannot be used as the value in an attribute assignment (SQLSTATE 429B9).

The DEFAULT keyword cannot be used as the value in an assignment for update on a nickname where the data source does not support DEFAULT syntax.

row-fullselect
A fullselect that returns a single row with the number of columns corresponding to the number of *column-name*s specified for assignment. The values are assigned to each corresponding *column-name*. If the result of the *row-fullselect* is no rows, then null values are assigned.

A *row-fullselect* may contain references to columns of the target table of the UPDATE statement. For each row that is updated, the value of such a column in an expression is the value of the column in the row before the row is updated. An error is returned if there is more than one row in the result (SQLSTATE 21000).

WHERE
Introduces a condition that indicates what rows are updated. You can omit the clause, give a search condition, or name a cursor. If the clause is omitted, all rows of the table, view or nickname are updated.

search-condition
Each *column-name* in the search condition, other than in a subquery, must name a column of the table, view or nickname. When the search condition includes a subquery in which the same table is the base object of both the UPDATE and the subquery, the subquery is completely evaluated before any rows are updated.

The search-condition is applied to each row of the table, view or nickname and the updated rows are those for which the result of the search-condition is true.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a row, and the results used in applying the search condition. In actuality, a subquery with no correlated references is executed only once, whereas a subquery with a correlated reference may have to be executed once for each row.

CURRENT OF *cursor-name*
Identifies the cursor to be used in the update operation. The *cursor-name* must identify a declared cursor, explained in "DECLARE CURSOR". The DECLARE CURSOR statement must precede the UPDATE statement in the program.

The specified table, view, or nickname must also be named in the FROM clause of the SELECT statement of the cursor, and the result table of the cursor must not be read-only. (For an explanation of read-only result tables, see "DECLARE CURSOR".)

When the UPDATE statement is executed, the cursor must be positioned on a row; that row is updated.

This form of UPDATE cannot be used (SQLSTATE 42828) if the cursor references:

- A view on which an INSTEAD OF UPDATE trigger is defined
- A view that includes an OLAP function in the select list of the fullselect that defines the view
- A view that is defined, either directly or indirectly, using the WITH ROW MOVEMENT clause

WITH
Specifies the isolation level at which the UPDATE statement is executed.

RR
Repeatable Read

RS
Read Stability

CS
Cursor Stability

UR
Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. The WITH clause has no effect on nicknames, which always use the default isolation level of the statement.

Rules:

- ***Triggers:*** UPDATE statements may cause triggers to be executed. A trigger may cause other statements to be executed, or may raise error conditions based on the update values. If an update operation on a view causes an INSTEAD OF trigger to fire, validity, referential integrity, and constraints will be checked against the updates that are performed in the trigger, and not against the view that caused the trigger to fire, or its underlying tables.
- ***Assignment***: Update values are assigned to columns according to specific assignment rules.

- ***Validity***: The updated row must conform to any constraints imposed on the table (or on the base table of the view) by any unique index on an updated column.

  If a view is used that is not defined using WITH CHECK OPTION, rows can be changed so that they no longer conform to the definition of the view. Such rows are updated in the base table of the view and no longer appear in the view.

  If a view is used that is defined using WITH CHECK OPTION, an updated row must conform to the definition of the view. For an explanation of the rules governing this situation, see "CREATE VIEW".
- ***Check Constraint***: Update value must satisfy the check-conditions of the check constraints defined on the table.

  An UPDATE to a table with check constraints defined has the constraint conditions for each column updated evaluated once for each row that is updated. When processing an UPDATE statement, only the check constraints referring to the updated columns are checked.
- ***Referential Integrity***: The value of the parent unique keys cannot be changed if the update rule is RESTRICT and there are one or more dependent rows. However, if the update rule is NO ACTION, parent unique keys can be updated as long as every child has a parent key by the time the update statement completes. A non-null update value of a foreign key must be equal to a value of the primary key of the parent table of the relationship.
- ***XML values:*** When an XML column value is updated, the new value must be a well-formed XML document (SQLSTATE 2200M).

Notes:

- If an update value violates any constraints, or if any other error occurs during the execution of the UPDATE statement, no rows are updated. The order in which multiple rows are updated is undefined.
- An update to a view defined using the WITH ROW MOVEMENT clause could cause a delete operation and an insert operation against the underlying tables of the view. For details, see the description of the CREATE VIEW statement.
- When an UPDATE statement completes execution, the value of SQLERRD(3) in the SQLCA is the number of rows that qualified for the update operation. In the context of an SQL procedure statement, the value can be retrieved using the ROW_COUNT variable of the GET DIAGNOSTICS statement. The SQLERRD(5) field contains the number of rows inserted, deleted, or updated by all activated triggers.
- Unless appropriate locks already exist, one or more exclusive locks are acquired by the execution of a successful UPDATE statement. Until the locks are released, the updated row can only be accessed by the application process that performed the update (except for applications using the Uncommitted Read isolation level). For further information on locking, see the descriptions of the COMMIT, ROLLBACK, and LOCK TABLE statements.
- If the URL value of a DATALINK column is updated, this is the same as deleting the old DATALINK value then inserting the new one. First, if the old value was linked to a file, that file is unlinked. Then, unless the linkage attributes of the DATALINK value are empty, the specified file is linked to that column. The only exception to this is that if the URL of the new DATALINK value is *identical* to the URL of the existing DATALINK value, there is no need to communicate with the associated Data Links Manager to unlink and relink the same file. In this situation, the overhead is entirely eliminated.

The comment value of a DATALINK column can be updated without relinking the file by specifying an empty string as the URL path (for example, as the *data-location* argument of the DLVALUE scalar function or by specifying the new value to be the same as the old value).

If a DATALINK column is updated with a null, it is the same as deleting the existing DATALINK value.

An error may occur when attempting to update a DATALINK value if the file server of either the existing value or the new value is no longer registered with the database server (SQLSTATE 55022).

- When updating the column distribution statistics for a typed table, the subtable that first introduced the column must be specified.
- Multiple attribute assignments on the same structured type column occur in the order specified in the SET clause and, within a parenthesized set clause, in left-to-right order.
- An attribute assignment invokes the mutator method for the attribute of the user-defined structured type. For example, the assignment `st..a1=x` has the same effect as using the mutator method in the assignment `st = st..a1(x)`.
- While a given column may be a target column in only one conventional assignment, a column may be a target column in multiple attribute assignments (but only if it is not also a target column in a conventional assignment).
- When an identity column defined as a distinct type is updated, the entire computation is done in the source type, and the result is cast to the distinct type before the value is actually assigned to the column. (There is no casting of the previous value to the source type prior to the computation.)
- To have DB2 generate a value on a SET statement for an identity column, use the DEFAULT keyword:

  ```
  SET NEW.EMPNO = DEFAULT
  ```

  In this example, NEW.EMPNO is defined as an identity column, and the value used to update this column is generated by DB2.
- For more information about consuming values of a generated sequence for an identity column, or about exceeding the maximum value for an identity column, see "INSERT".
- With partitioned tables, an UPDATE WHERE CURRENT OF *cursor-name* operation can move a row from one data partition to another. After this occurs, the cursor is no longer positioned on the row, and no further UPDATE WHERE CURRENT OF *cursor-name* modifications to that row are possible. The next row in the cursor can be fetched, however.

Examples:

- *Example 1:* Change the job (JOB) of employee number (EMPNO) '000290' in the EMPLOYEE table to 'LABORER'.

  ```
  UPDATE EMPLOYEE
     SET JOB = 'LABORER'
     WHERE EMPNO = '000290'
  ```
- *Example 2:* Increase the project staffing (PRSTAFF) by 1.5 for all projects that department (DEPTNO) 'D21' is responsible for in the PROJECT table.

  ```
  UPDATE PROJECT
     SET PRSTAFF = PRSTAFF + 1.5
     WHERE DEPTNO = 'D21'
  ```

- *Example 3:* All the employees except the manager of department (WORKDEPT) 'E21' have been temporarily reassigned. Indicate this by changing their job (JOB) to NULL and their pay (SALARY, BONUS, COMM) values to zero in the EMPLOYEE table.

  ```
  UPDATE EMPLOYEE
    SET JOB=NULL, SALARY=0, BONUS=0, COMM=0
    WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
  ```

  This statement could also be written as follows.

  ```
  UPDATE EMPLOYEE
    SET (JOB, SALARY, BONUS, COMM) = (NULL, 0, 0, 0)
    WHERE WORKDEPT = 'E21' AND JOB <> 'MANAGER'
  ```

- *Example 4:* Update the salary and the commission column of the employee with employee number 000120 to the average of the salary and of the commission of the employees of the updated row's department, respectively.

  ```
  UPDATE (SELECT SALARY,
      COMM,
      AVG(SALARY) OVER (PARTITION BY WORKDEPT),
      AVG(COMM) OVER (PARTITION BY WORKDEPT)
    FROM EMPLOYEE)
    AS E(SALARY, COMM, AVGSAL, AVGCOMM)
    SET (SALARY, COMM)
      = (AVGSAL, AVGCOMM)
    WHERE EU.EMPNO = '000120'
  ```

  The previous statement is semantically equivalent to the following statement, but requires only one access to the EMPLOYEE table, whereas the following statement specifies the EMPLOYEE table twice.

  ```
  UPDATE EMPLOYEE EU
    SET (EU.SALARY, EU.COMM)
    =
  (SELECT AVG(ES.SALARY), AVG(ES.COMM)
    FROM EMPLOYEE ES
    WHERE ES.WORKDEPT = EU.WORKDEPT)
    WHERE EU.EMPNO = '000120'
  ```

- *Example 5:* In a C program display the rows from the EMPLOYEE table and then, if requested to do so, change the job (JOB) of certain employees to the new job keyed in.

  ```
  EXEC SQL  DECLARE C1 CURSOR FOR
                 SELECT *
                   FROM EMPLOYEE
                   FOR UPDATE OF JOB;

  EXEC SQL  OPEN C1;

  EXEC SQL  FETCH C1 INTO ...     ;
  if ( strcmp (change, "YES") == 0 )
    EXEC SQL  UPDATE EMPLOYEE
                SET JOB = :newjob
                WHERE CURRENT OF C1;

  EXEC SQL  CLOSE C1;
  ```

- *Example 6:* These examples mutate attributes of column objects.

  Assume that the following types and tables exist:

  ```
  CREATE TYPE POINT AS (X INTEGER, Y INTEGER)
    NOT FINAL WITHOUT COMPARISONS
    MODE DB2SQL
  CREATE TYPE CIRCLE AS (RADIUS INTEGER, CENTER POINT)
    NOT FINAL WITHOUT COMPARISONS
    MODE DB2SQL
  ```

```
CREATE TABLE CIRCLES (ID INTEGER, OWNER VARCHAR(50), C CIRCLE
```

The following example updates the CIRCLES table by changing the OWNER column and the RADIUS attribute of the CIRCLE column where the ID is 999:

```
UPDATE CIRCLES
  SET OWNER = 'Bruce'
    C..RADIUS = 5
  WHERE ID = 999
```

The following example transposes the X and Y coordinates of the center of the circle identified by 999:

```
UPDATE CIRCLES
  SET C..CENTER..X = C..CENTER..Y,
    C..CENTER..Y = C..CENTER..X
  WHERE ID = 999
```

The following example is another way of writing both of the above statements. This example combines the effects of both of the above examples:

```
UPDATE CIRCLES
  SET (OWNER,C..RADIUS,C..CENTER..X,C..CENTER..Y) =
    ('Bruce',5,C..CENTER..Y,C..CENTER..X)
  WHERE ID = 999
```

- *Example 7:* Update the XMLDOC column of the DOCUMENTS table with DOCID ′001′ to the character string that is selected and parsed from the XMLTEXT table.

```
UPDATE DOCUMENTS SET XMLDOC =
  (SELECT XMLPARSE(DOCUMENT C1 STRIP WHITESPACE)
    FROM XMLTEXT WHERE TEXTID = '001')
WHERE DOCID = '001'
```

Related reference:
- "Assignments and comparisons" in *SQL Reference, Volume 1*
- "Expressions" in *SQL Reference, Volume 1*
- "Search conditions" in *SQL Reference, Volume 1*
- "SQLCA (SQL communications area)" on page 1357
- "Subselect" on page 1211
- "ALTER TABLE " on page 854
- "CREATE VIEW " on page 1034
- "DECLARE CURSOR statement" in *SQL Reference, Volume 2*
- "INSERT " on page 1109
- "MERGE statement" in *SQL Reference, Volume 2*

Related samples:
- "dbinline.sqc -- How to use inline SQL Procedure Language (C)"
- "spserver.sqc -- Definition of various types of stored procedures (C)"
- "tbmod.sqc -- How to modify table data (C)"
- "dtstruct.sqC -- Create, use, drop a hierarchy of structured types and typed tables (C++)"
- "spserver.sqC -- Definition of various types of stored procedures (C++)"
- "tbmod.sqC -- How to modify table data (C++)"
- "SpServer.java -- Provide a variety of types of stored procedures to be called from (JDBC)"
- "TbMod.java -- How to modify table data (JDBC)"
- "SpServer.sqlj -- Provide a variety of types of stored procedures to be called from (SQLj)"

- “TbMod.sqlj -- How to modify table data (SQLj)”
- “tbmod.c -- How to modify table data”
- “updat.sqb -- How to update, delete and insert table data (MF COBOL)”
- “varinp.sqb -- How to update table data using parameter markers (MF COBOL)”

Chapter 41. SQL Statements for Users

COMMIT

The COMMIT statement terminates a unit of work and commits the database changes that were made by that unit of work.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

None required.

Syntax:

```
             ┌WORK┐
►►──COMMIT───┴────┴──────────────────────────────────────────────►◄
```

Description:
The unit of work in which the COMMIT statement is executed is terminated and a new unit of work is initiated. All changes made by the following statements executed during the unit of work are committed: ALTER, COMMENT, CREATE, DROP, GRANT, LOCK TABLE, REVOKE, SET INTEGRITY, SET Variable, and the data change statements (INSERT, DELETE, MERGE, UPDATE), including those nested in a query.

The following statements, however, are not under transaction control and changes made by them are independent of the COMMIT statement:
- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT LOCK TIMEOUT
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET EVENT MONITOR STATE
- SET PASSTHRU

  Note: Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.
- SET PATH
- SET SCHEMA
- SET SERVER OPTION

All locks acquired by the unit of work subsequent to its initiation are released, except necessary locks for open cursors that are declared WITH HOLD. All open cursors not defined WITH HOLD are closed. Open cursors defined WITH HOLD remain open, and the cursor is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.) All LOB locators are freed. Note that this is true even when the locators are associated with LOB values retrieved via a cursor that has the WITH HOLD property.

All savepoints set within the transaction are released.

Notes:
- It is strongly recommended that each application process explicitly ends its unit of work before terminating. If the application program ends normally without a COMMIT or ROLLBACK statement then the database manager attempts a commit or rollback depending on the application environment.
- For information on the impact of COMMIT on cached dynamic SQL statements, see “EXECUTE”.
- For information on potential impacts of COMMIT on declared temporary tables, see “DECLARE GLOBAL TEMPORARY TABLE”.

Example:

Commit alterations to the database made since the last commit point.

```
COMMIT WORK
```

Related reference:
- “DECLARE GLOBAL TEMPORARY TABLE statement” in *SQL Reference, Volume 2*
- “EXECUTE statement” in *SQL Reference, Volume 2*

Related samples:

- "dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)"
- "tbconstr.sqc -- How to create, use, and drop constraints (C)"
- "tbsavept.sqc -- How to use external savepoints (C)"
- "tbconstr.sqC -- How to create, use, and drop constraints (C++)"

CONNECT (Type 1)

The CONNECT (Type 1) statement connects an application process to the identified application server according to the rules for remote unit of work.

An application process can only be connected to one application server at a time. This is called the *current server*. A default application server may be established when the application requester is initialized. If implicit connect is available and an application process is started, it is implicitly connected to the default application server. The application process can explicitly connect to a different application server by issuing a CONNECT TO statement. A connection lasts until a CONNECT RESET statement or a DISCONNECT statement is issued or until another CONNECT TO statement changes the application server.

Invocation:

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization:

The authorization ID of the statement must be authorized to connect to the identified application server. Depending on the authentication setting for the database, the authorization check might be performed by either the client or the server. For a partitioned database, the user and group definitions must be identical across database partitions.

Syntax:

```
►►──CONNECT──────────────────────────────────────────────────────────►

►──┬──────────────────────────────────────────────────────────────┬──►◄
   ├─TO──┬─server-name───┬──┬────────────────┬──┬─────────────────┬┤
   │     └─host-variable─┘  └─┤ lock-block ├─┘  └─┤ authorization ├─┘│
   ├─RESET──────────────────────────────────────────────────────────┤
   │                   (1)                                          │
   └─┤ authorization ├──────────────────────────────────────────────┘
```

authorization:

```
├──USER──┬─authorization-name─┬──USING──┬─password──────┬──────────────►
         └─host-variable──────┘         └─host-variable─┘
```

```
►─┬─────────────────────────────────────────────────┬─────────────────────────────────────┤
  └─NEW──┬─password──────┬──CONFIRM──password─┘
         └─host-variable─┘
```

lock-block:

```
   ┌─IN SHARE MODE──────────────────────────────────┐
├──┼────────────────────────────────────────────────┼──────────────────────────────┤
   └─IN EXCLUSIVE MODE──┬─────────────────────────┬─┘
                        └─ON SINGLE DBPARTITIONNUM─┘
```

Notes:

1 This form is only valid if implicit connect is enabled.

Description:

CONNECT (with no operand)
Returns information about the current server. The information is returned in the SQLERRP field of the SQLCA as described in "Successful Connection".

If a connection state exists, the authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If the authorization ID is longer than 8 bytes, it will be truncated to 8 bytes, and the truncation will be flagged in the SQLWARN0 and SQLWARN1 fields of the SQLCA, with 'W' and 'A', respectively. If the database configuration parameter DYN_QUERY_MGMT is enabled, then the SQLWARN0 and SQLWARN7 fields of the SQLCA will be flagged with 'W' and 'E', respectively.

If no connection exists and implicit connect is possible, then an attempt to make an implicit connection is made. If implicit connect is not available, this attempt results in an error (no existing connection). If no connection, then the SQLERRMC field is blank.

The territory code and code page of the application server are placed in the SQLERRMC field (as they are with a successful CONNECT TO statement).

This form of CONNECT:

- Does not require the application process to be in the connectable state.
- If connected, does not change the connection state.
- If unconnected and implicit connect is available, a connection to the default application server is made. In this case, the country or region code and code page of the application server are placed in the SQLERRMC field, like a successful CONNECT TO statement.
- If unconnected and implicit connect is not available, the application process remains unconnected.
- Does not close cursors.

TO *server-name* or *host-variable*
Identifies the application server by the specified *server-name* or a *host-variable* which contains the server-name.

If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-justified and must not be delimited by quotation marks.

Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

Note: DB2 UDB for OS/390 and z/OS supports a 16-byte location name, and DB2 UDB for iSeries supports an 18-byte target database name. DB2 Version 8 only supports the use of an 8-byte database alias name on the SQL CONNECT statement. However, the database alias name can be mapped to an 18-byte database name through the Database Connection Service Directory.

When the CONNECT TO statement is executed, the application process must be in the connectable state.

Successful Connection:

If the CONNECT TO statement is successful:

- All open cursors are closed, all prepared statements are destroyed, and all locks are released from the previous application server.
- The application process is disconnected from its previous application server, if any, and connected to the identified application server.
- The actual name of the application server (not an alias) is placed in the CURRENT SERVER special register.
- Information about the application server is placed in the SQLERRP field of the SQLCA. If the application server is an IBM product, the information has the form *pppvvrrm*, where:
  - *ppp* identifies the product as follows:
    - DSN for DB2 UDB for OS/390 and z/OS
    - ARI for DB2 Server for VSE & VM
    - QSQ for DB2 UDB for iSeries
    - SQL for DB2 Database for Linux, UNIX, and Windows
  - *vv* is a two-digit version identifier, such as '08'
  - *rr* is a two-digit release identifier, such as '01'
  - *m* is a one-digit modification level identifier, such as '0'.

  This release (Version 9) of DB2 Database for Linux, UNIX, and Windows is identified as 'SQL09010'.
- The SQLERRMC field of the SQLCA is set to contain the following values (separated by X'FF')
  1. the country or region code of the application server (or blanks if using DB2 Connect),
  2. the code page of the application server (or CCSID if using DB2 Connect),
  3. the authorization ID (up to first 8 bytes only),
  4. the database alias,
  5. the platform type of the application server. Currently identified values are:

Token	Server
QAS	DB2 Universal Database for iSeries
QDB2	DB2 Universal Database for OS/390 and z/OS
QDB2/6000	DB2 Database for AIX
QDB2/HPUX	DB2 Database for HP-UX

QDB2/LINUX	DB2 Database for Linux
QDB2/NT	DB2 Database for Windows
QDB2/SUN	DB2 Database for Solaris Operating System
QSQLDS/VM	DB2 Server for VM
QSQLDS/VSE	DB2 Server for VSE

6. The agent ID. It identifies the agent executing within the database manager on behalf of the application. This field is the same as the agent_id element returned by the database monitor.
7. The agent index. It identifies the index of the agent and is used for service.
8. Database partition number. For a non-partitioned database, this is always 0, if present.
9. The code page of the application client.
10. Number of database partitions in a partitioned database. If the database cannot be distributed, the value is 0 (zero). Token is present only with Version 5 or later.

- The SQLERRD(1) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
- The SQLERRD(2) field of the SQLCA indicates the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction.
- The SQLERRD(3) field of the SQLCA indicates whether or not the database on the connection is updatable. A database is initially updatable, but is changed to read-only if a unit of work determines the authorization ID cannot perform updates. The value is one of:
  - 1 - updatable
  - 2 - read-only
- The SQLERRD(4) field of the SQLCA returns certain characteristics of the connection. The value is one of:

0	N/A (only possible if running from a down-level client that is one-phase commit and is an updater).
1	one-phase commit.
2	one-phase commit; read-only (only applicable to connections to DRDA1 databases in a TP Monitor environment).
3	two-phase commit.

- The SQLERRD(5) field of the SQLCA returns the authentication type for the connection. The value is one of:

0	Authenticated on the server.
1	Authenticated on the client.
2	Authenticated using DB2 Connect.
4	Authenticated on the server with encryption.

5	Authenticated using DB2 Connect with encryption.
7	Authenticated using an external Kerberos security mechanism.
9	Authenticated using an external GSS API plug-in security mechanism.
11	Authenticated on the server, which accepts encrypted data.
255	Authentication not specified.

- The SQLERRD(6) field of the SQLCA returns the database partition number of the database partition to which the connection was made if the database is distributed. Otherwise, a value of 0 is returned.
- The SQLWARN1 field in the SQLCA will be set to 'A' if the authorization ID of the successful connection is longer than 8 bytes. This indicates that truncation has occurred. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.
- The SQLWARN7 field in the SQLCA will be set to 'E' if the database configuration parameter DYN_QUERY_MGMT for the database is enabled. The SQLWARN0 field in the SQLCA will be set to 'W' to indicate this warning.

Unsuccessful Connection:

If the CONNECT TO statement is unsuccessful:

- The SQLERRP field of the SQLCA is set to the name of the module at the application requester that detected the error. The first three characters of the module name identify the product.
- If the CONNECT TO statement is unsuccessful because the application process is not in the connectable state, the connection state of the application process is unchanged.
- If the CONNECT TO statement is unsuccessful because the *server-name* is not listed in the local directory, an error message (SQLSTATE 08001) is issued and the connection state of the application process remains unchanged:
  - If the application requester was not connected to an application server then the application process remains unconnected.
  - If the application requester was already connected to an application server, the application process remains connected to that application server. Any further statements are executed at that application server.
- If the CONNECT TO statement is unsuccessful for any other reason, the application process is placed into the unconnected state.

IN SHARE MODE
Allows other concurrent connections to the database and prevents other users from connecting to the database in exclusive mode.

IN EXCLUSIVE MODE
Prevents concurrent application processes from executing any operations at the application server, unless they have the same authorization ID as the user holding the exclusive lock. This option is not supported by DB2 Connect.

ON SINGLE DBPARTITIONNUM
Specifies that the coordinator database partition is connected in exclusive mode and all other database partitions are connected in share mode. This option is only effective in a partitioned database.

RESET
Disconnects the application process from the current server. A commit

operation is performed. If implicit connect is available, the application process remains unconnected until an SQL statement is issued.

USER *authorization-name/host-variable*
: Identifies the user ID trying to connect to the application server. If a host-variable is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The user ID that is contained within the *host-variable* must be left justified and must not be delimited by quotation marks.

USING *password/host-variable*
: Identifies the password of the user ID trying to connect to the application server. The *password* or *host-variable* can be up to 14 bytes long. If a host variable is specified, it must be a character string variable with a length attribute not greater than 14, and it must not include an indicator variable.

NEW *password/host-variable* **CONFIRM** *password*
: Identifies the new password that should be assigned to the user ID identified by the USER option. The *password* or *host-variable* can be up to 14 bytes long. If a host variable is specified, it must be a character string variable with a length attribute not greater than 14, and it must not include an indicator variable. The system on which the password will be changed depends on how the user authentication has been set up.

Notes:

- It is good practice for the first SQL statement executed by an application process to be the CONNECT TO statement.
- If a CONNECT TO statement is issued to the current application server with a different user ID and password then the conversation is deallocated and reallocated. All cursors are closed by the database manager (with the loss of the cursor position if the WITH HOLD option was used).
- If a CONNECT TO statement is issued to the current application server with the same user ID and password then the conversation is not deallocated and reallocated. Cursors, in this case, are not closed.
- To use a multiple-partition partitioned database environment, the user or application must connect to one of the database partitions listed in the `db2nodes.cfg` file. You should try to ensure that not all users use the same database partition as the coordinator partition.
- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft® Windows NT Security Account Manager (SAM)-compatible name. The qualifier must be a NetBIOS style name, which has a maximum length of 15 bytes. For example, 'Domain\User'.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODE can be specified in place of DBPARTITIONNUM

Examples:

Example 1: In a C program, connect to the application server TOROLAB, using database alias TOROLAB, user ID FERMAT, and password THEOREM.

```
EXEC SQL  CONNECT TO TOROLAB USER FERMAT USING THEOREM;
```

Example 2: In a C program, connect to an application server whose database alias is stored in the host variable APP_SERVER (varchar(8)). Following a successful connection, copy the 3-character product identifier of the application server to the variable PRODUCT (char(3)).

```
EXEC SQL  CONNECT TO :APP_SERVER;
if (strncmp(SQLSTATE,'00000',5))
  strncpy(PRODUCT,sqlca.sqlerrp,3);
```

Related concepts:

- Chapter 9, "Database partitioning across multiple database partitions," on page 49
- "Distributed relational databases" in *SQL Reference, Volume 1*

Related samples:

- "advsql.sqb -- How to read table data using CASE (MF COBOL)"
- "dbmcon.sqc -- How to use multiple databases (C)"
- "dbmcon.sqC -- How to use multiple databases (C++)"

CONNECT (Type 2)

The CONNECT (Type 2) statement connects an application process to the identified application server and establishes the rules for application-directed distributed unit of work. This server is then the current server for the process.

Most aspects of a CONNECT (Type 1) statement also apply to a CONNECT (Type 2) statement. Rather than repeating that material here, this section describes only those elements of Type 2 that differ from Type 1.

Invocation:

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization:

The authorization ID of the statement must be authorized to connect to the identified application server. Depending on the authentication setting for the database, the authorization check might be performed by either the client or the server. For a partitioned database, the user and group definitions must be identical across database partitions.

Syntax:

The selection between Type 1 and Type 2 is determined by precompiler options. For an overview of these options, see "Distributed relational databases".

```
►►──CONNECT──────────────────────────────────────────────────────────►
```

CONNECT (Type 2)

```
►►─┬───────────────────────────────────────────────────────────────────┬──────►◄
   ├─TO──┬─server-name───┬──┬──────────────┬──┬─────────────────┬──────┤
   │     └─host-variable─┘  └─┤ lock-block ├─┘  └─┤ authorization ├─┘      │
   ├─RESET─────────────────────────────────────────────────────────────┤
   │                      (1)                                          │
   └─┤ authorization ├─────────────────────────────────────────────────┘
```

authorization:

```
├──USER──┬─authorization-name─┬──USING──┬─password──────┬──────────────────►
         └─host-variable──────┘         └─host-variable─┘

►──┬──────────────────────────────────────────────────┬───────────────────┤
   └─NEW──┬─password──────┬──CONFIRM──password─┘
          └─host-variable─┘
```

lock-block:

```
   ┌─IN SHARE MODE───────────────────────────────────────┐
├──┴─IN EXCLUSIVE MODE──┬───────────────────────────┬────┴──────────────────┤
                        └─ON SINGLE DBPARTITIONNUM──┘
```

Notes:

1 This form is only valid if implicit connect is enabled.

Description:

TO *server-name/host-variable*
The rules for coding the name of the server are the same as for Type 1.

If the SQLRULES(STD) option is in effect, the *server-name* must not identify an existing connection of the application process, otherwise an error (SQLSTATE 08002) is raised.

If the SQLRULES(DB2) option is in effect and the *server-name* identifies an existing connection of the application process, that connection is made current and the old connection is placed into the dormant state. That is, the effect of the CONNECT statement in this situation is the same as that of a SET CONNECTION statement.

For information about the specification of SQLRULES, see "Options that Govern Distributed Unit of Work Semantics".

Successful Connection

If the CONNECT TO statement is successful:

- A connection to the application server is either created (or made non-dormant) and placed into the current and held states.
- If the CONNECT TO is directed to a different server than the current server, then the current connection is placed into the dormant state.
- The CURRENT SERVER special register and the SQLCA are updated in the same way as for CONNECT (Type 1).

Unsuccessful Connection

If the CONNECT TO statement is unsuccessful:

- No matter what the reason for failure, the connection state of the application process and the states of its connections are unchanged.
- As with an unsuccessful Type 1 CONNECT, the SQLERRP field of the SQLCA is set to the name of the module at the application requester or server that detected the error.

CONNECT (with no operand)**, IN SHARE/EXCLUSIVE MODE, USER,** and **USING**

If a connection exists, Type 2 behaves like a Type 1. The authorization ID and database alias are placed in the SQLERRMC field of the SQLCA. If a connection does not exist, no attempt to make an implicit connection is made and the SQLERRP and SQLERRMC fields return a blank. (Applications can check if a current connection exists by checking these fields.)

A CONNECT with no operand that includes USER and USING can still connect an application process to a database using the DB2DBDFT environment variable. This method is equivalent to a Type 2 CONNECT RESET, but permits the use of a user ID and password.

RESET

Equivalent to an explicit connect to the default database if it is available. If a default database is not available, the connection state of the application process and the states of its connections are unchanged.

Availability of a default database is determined by installation options, environment variables, and authentication settings.

Rules:

- As outlined in "Options that Govern Distributed Unit of Work Semantics", a set of connection options governs the semantics of connection management. Default values are assigned to every preprocessed source file. An application can consist of multiple source files precompiled with different connection options.

  Unless a SET CLIENT command or API has been executed first, the connection options used when preprocessing the source file containing the first SQL statement executed at run time become the effective connection options.

  If a CONNECT statement from a source file preprocessed with different connection options is subsequently executed without the execution of any intervening SET CLIENT command or API, an error (SQLSTATE 08001) is returned. Note that once a SET CLIENT command or API has been executed, the connection options used when preprocessing all source files in the application are ignored.

  Example 1 in the "Examples" section of this statement illustrates these rules.
- Although the CONNECT TO statement can be used to establish or switch connections, CONNECT TO with the USER/USING clause will only be accepted when there is no current or dormant connection to the named server. The connection must be released before issuing a connection to the same server with the USER/USING clause, otherwise it will be rejected (SQLSTATE 51022). Release the connection by issuing a DISCONNECT statement or a RELEASE statement followed by a COMMIT statement.

Notes:

- Implicit connect is supported for the first SQL statement in an application with Type 2 connections. In order to execute SQL statements on the default database, first the CONNECT RESET or the CONNECT USER/USING statement must be used to establish the connection. The CONNECT statement with no operands will display information about the current connection if there is one, but will not connect to the default database if there is no current connection.

- The *authorization-name* SYSTEM cannot be explicitly specified in the CONNECT statement. However, on Windows operating systems, local applications running under the Local System Account can implicitly connect to the database, such that the user ID is SYSTEM.
- When connecting to Windows Server explicitly, the *authorization-name* or user *host-variable* can be specified using the Microsoft Windows Security Account Manager (SAM)-compatible name. The qualifier must be a NetBIOS style name, which has a maximum length of 15 bytes. For example, 'Domain\User'.

Comparing Type 1 and Type 2 CONNECT Statements:

The semantics of the CONNECT statement are determined by the CONNECT precompiler option or the SET CLIENT API (see "Options that Govern Distributed Unit of Work Semantics"). CONNECT Type 1 or CONNECT Type 2 can be specified and the CONNECT statements in those programs are known as Type 1 and Type 2 CONNECT statements, respectively. Their semantics are described below:

Use of **CONNECT TO**:

Type 1	Type 2
Each unit of work can only establish connection to one application server.	Each unit of work can establish connection to multiple application servers.
The current unit of work must be committed or rolled back before allowing a connection to another application server.	The current unit of work need not be committed or rolled back before connecting to another application server.
The CONNECT statement establishes the current connection. Subsequent SQL requests are forwarded to this connection until changed by another CONNECT.	Same as Type 1 CONNECT if establishing the first connection. If switching to a dormant connection and SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to the current connection is valid and does not change the current connection.	Same as Type 1 CONNECT if the SQLRULES precompiler option is set to DB2. If SQLRULES is set to STD, then the SET CONNECTION statement must be used instead.
Connecting to another application server disconnects the current connection. The new connection becomes the current connection. Only one connection is maintained in a unit of work.	Connecting to another application server puts the current connection into the *dormant state*. The new connection becomes the current connection. Multiple connections can be maintained in a unit of work. If the CONNECT is for an application server on a dormant connection, it becomes the current connection. Connecting to a dormant connection using CONNECT is only allowed if SQLRULES(DB2) was specified. If SQLRULES(STD) was specified, then the SET CONNECTION statement must be used instead.

Type 1	Type 2
SET CONNECTION statement is supported for Type 1 connections, but the only valid target is the current connection.	SET CONNECTION statement is supported for Type 2 connections to change the state of a connection from dormant to current.

Use of **CONNECT...USER...USING**:

Type 1	Type 2
Connecting with the USER...USING clauses disconnects the current connection and establishes a new connection with the given authorization name and password.	Connecting with the USER/USING clause will only be accepted when there is no current or dormant connection to the same named server.

Use of **Implicit CONNECT, CONNECT RESET**, and **Disconnecting**:

Type 1	Type 2
CONNECT RESET can be used to disconnect the current connection.	CONNECT RESET is equivalent to connecting to the default application server explicitly if one has been defined in the system. Connections can be disconnected by the application at a successful COMMIT. Prior to the commit, use the RELEASE statement to mark a connection as release-pending. All such connections will be disconnected at the next COMMIT. An alternative is to use the precompiler options DISCONNECT(EXPLICIT), DISCONNECT(CONDITIONAL), DISCONNECT(AUTOMATIC), or the DISCONNECT statement instead of the RELEASE statement.
After using CONNECT RESET to disconnect the current connection, if the next SQL statement is not a CONNECT statement, then it will perform an implicit connect to the default application server if one has been defined in the system.	CONNECT RESET is equivalent to an explicit connect to the default application server if one has been defined in the system.
It is an error to issue consecutive CONNECT RESETs.	It is an error to issue consecutive CONNECT RESETs ONLY if SQLRULES(STD) was specified because this option disallows the use of CONNECT to existing connection.
CONNECT RESET also implicitly commits the current unit of work.	CONNECT RESET does not commit the current unit of work.
If an existing connection is disconnected by the system for whatever reasons, then subsequent non-CONNECT SQL statements to this database will receive an SQLSTATE of 08003.	If an existing connection is disconnected by the system, COMMIT, ROLLBACK, and SET CONNECTION statements are still permitted.
The unit of work will be implicitly committed when the application process terminates successfully.	Same as Type 1.

Type 1	Type 2
All connections (only one) are disconnected when the application process terminates.	All connections (current, dormant, and those marked for release pending) are disconnected when the application process terminates.

CONNECT Failures:

Type 1	Type 2
Regardless of whether there is a current connection when a CONNECT fails (with an error other than server-name not defined in the local directory), the application process is placed in the unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.	If there is a current connection when a CONNECT fails, the current connection is unaffected. If there was no current connection when the CONNECT fails, then the program is then in an unconnected state. Subsequent non-CONNECT statements receive an SQLSTATE of 08003.

Examples:

Example 1:

This example illustrates the use of multiple source programs (shown in the boxes), some preprocessed with different connection options (shown above the code), and one of which contains a SET CLIENT API call.

PGM1: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO OTTAWA;
exec sql SELECT col1 INTO :hv1
FROM tbl1;
...
```

PGM2: CONNECT(2) SQLRULES(STD) DISCONNECT(AUTOMATIC)

```
...
exec sql CONNECT TO QUEBEC;
exec sql SELECT col1 INTO :hv1
FROM tbl2;
...
```

PGM3: CONNECT(2) SQLRULES(STD) DISCONNECT(EXPLICIT)

```
...
SET CLIENT CONNECT 2  SQLRULES DB2  DISCONNECT EXPLICIT 1
exec sql CONNECT TO LONDON;
exec sql SELECT col1 INTO :hv1
FROM tbl3;
...
```

1 Note: not the actual syntax of the SET CLIENT API

PGM4: CONNECT(2) SQLRULES(DB2) DISCONNECT(CONDITIONAL)

```
...
exec sql CONNECT TO REGINA;
exec sql SELECT col1 INTO :hv1
FROM tbl4;
...
```

If the application executes PGM1 then PGM2:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to QUEBEC fails with SQLSTATE 08001 because both SQLRULES and DISCONNECT are different.

If the application executes PGM1 then PGM3:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to LONDON runs: connect=2, sqlrules=DB2, disconnect=EXPLICIT

This is OK because the SET CLIENT API is run before the second CONNECT statement.

If the application executes PGM1 then PGM4:

- connect to OTTAWA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL
- connect to REGINA runs: connect=2, sqlrules=DB2, disconnect=CONDITIONAL

This is OK because the preprocessor options for PGM1 are the same as those for PGM4.

Example 2:

This example shows the interrelationships of the CONNECT (Type 2), SET CONNECTION, RELEASE, and DISCONNECT statements. S0, S1, S2, and S3 represent four servers.

Sequence	Statement	Current Server	Dormant Connections	Release Pending
0	No statement	None	None	None
1	SELECT * FROM TBLA	S0 (default)	None	None
2	CONNECT TO S1	S1	S0	None
	SELECT * FROM TBLB	S1	S0	None
3	CONNECT TO S2	S2	S0, S1	None
	UPDATE TBLC SET ...	S2	S0, S1	None
4	CONNECT TO S3	S3	S0, S1, S2	None
	SELECT * FROM TBLD	S3	S0, S1, S2	None
5	SET CONNECTION S2	S2	S0, S1, S3	None
6	RELEASE S3	S2	S0, S1	S3
7	COMMIT	S2	S0, S1	None
8	SELECT * FROM TBLE	S2	S0, S1	None
9	DISCONNECT S1	S2	S0	None
	SELECT * FROM TBLF	S2	S0	None

Related concepts:

- “Distributed relational databases” in *SQL Reference, Volume 1*

Related reference:

- “CONNECT (Type 1) ” on page 1159

Related samples:
- "dbmcon.sqc -- How to use multiple databases (C)"
- "dbmcon.sqC -- How to use multiple databases (C++)"

DISCONNECT

The DISCONNECT statement destroys one or more connections when there is no active unit of work (that is, after a commit or rollback operation). If a single connection is the target of the DISCONNECT statement, the connection is destroyed only if the database has participated in an existing unit of work, regardless of whether there is an active unit of work. For example, if several other databases have done work, but the target in question has not, it can still be disconnected without destroying the connection.

Invocation:

Although an interactive SQL facility might provide an interface that gives the appearance of interactive execution, this statement can only be embedded within an application program. It is an executable statement that cannot be dynamically prepared.

Authorization:

None required.

Syntax:

```
                            (1)
►►──DISCONNECT──┬─server-name───┬────────────────────────────────────►◄
                ├─host-variable─┤
                ├─CURRENT───────┤
                │     ┌─SQL─┐   │
                └─ALL─┴─────┴───┘
```

Notes:

1 Note that an application server named CURRENT or ALL can only be identified by a host variable.

Description:

server-name or *host-variable*
: Identifies the application server by the specified *server-name* or a *host-variable* which contains the *server-name*.

  If a *host-variable* is specified, it must be a character string variable with a length attribute that is not greater than 8, and it must not include an indicator variable. The *server-name* that is contained within the *host-variable* must be left-justified and must not be delimited by quotation marks.

  Note that the *server-name* is a database alias identifying the application server. It must be listed in the application requester's local directory.

  The specified database-alias or the database-alias contained in the host variable must identify an existing connection of the application process. If the database-alias does not identify an existing connection, an error (SQLSTATE 08003) is raised.

CURRENT
: Identifies the current connection of the application process. The application process must be in the connected state. If not, an error (SQLSTATE 08003) is raised.

ALL
: Indicates that all existing connections of the application process are to be destroyed. An error or warning does not occur if no connections exist when the statement is executed. The optional keyword SQL is included to be consistent with the syntax of the RELEASE statement.

Rules:

- Generally, the DISCONNECT statement cannot be executed while within a unit of work. If attempted, an error (SQLSTATE 25000) is raised. The exception to this rule is if a single connection is specified to be disconnected and the database has not participated in an existing unit of work. In this case, it does not matter if there is an active unit of work when the DISCONNECT statement is issued.
- The DISCONNECT statement cannot be executed at all in the Transaction Processing (TP) Monitor environment (SQLSTATE 25000). It is used when the SYNCPOINT precompiler option is set to TWOPHASE.

Notes:

- If the DISCONNECT statement is successful, each identified connection is destroyed.

  If the DISCONNECT statement is unsuccessful, the connection state of the application process and the states of its connections are unchanged.
- If DISCONNECT is used to destroy the current connection, the next executed SQL statement should be CONNECT or SET CONNECTION.
- Type 1 CONNECT semantics do not preclude the use of DISCONNECT. However, though DISCONNECT CURRENT and DISCONNECT ALL can be used, they will not result in a commit operation like a CONNECT RESET statement would do.

  If *server-name* or *host-variable* is specified in the DISCONNECT statement, it must identify the current connection because Type 1 CONNECT only supports one connection at a time. Generally, DISCONNECT will fail if within a unit of work with the exception noted in “Rules”.
- Resources are required to create and maintain remote connections. Thus, a remote connection that is not going to be reused should be destroyed as soon as possible.
- Connections can also be destroyed during a commit operation because the connection option is in effect. The connection option could be AUTOMATIC, CONDITIONAL, or EXPLICIT, which can be set as a precompiler option or through the SET CLIENT API at run time. For information about the specification of the DISCONNECT option, see “Distributed relational databases”.

Examples:

Example 1: The SQL connection to IBMSTHDB is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

```
EXEC SQL DISCONNECT IBMSTHDB;
```

Example 2: The current connection is no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy the connection.

```
EXEC SQL DISCONNECT CURRENT;
```

Example 3: The existing connections are no longer needed by the application. The following statement should be executed after a commit or rollback operation to destroy all the connections.

```
EXEC SQL DISCONNECT ALL;
```

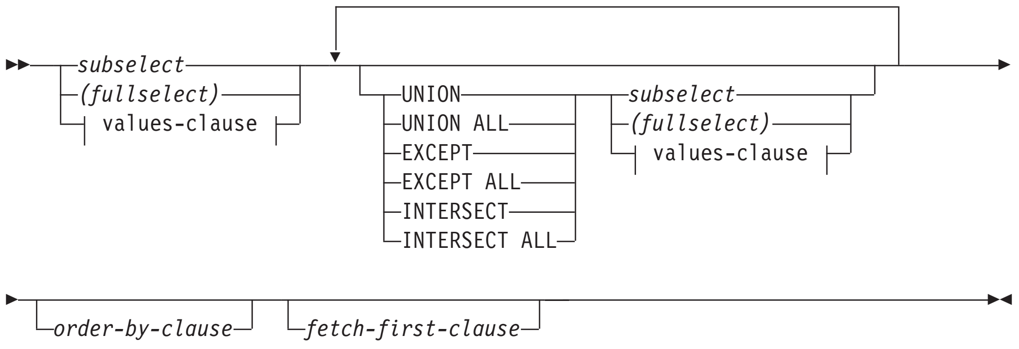
Related concepts:

- "Distributed relational databases" in *SQL Reference, Volume 1*

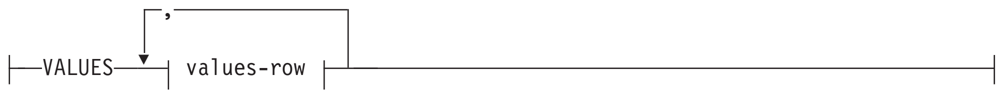
Related samples:

- "dbconn.sqc -- How to connect to and disconnect from a database (C)"
- "dbmcon.sqc -- How to use multiple databases (C)"
- "dbconn.sqC -- How to connect to and disconnect from a database (C++)"
- "dbmcon.sqC -- How to use multiple databases (C++)"
- "Util.java -- Utilities for JDBC sample programs (JDBC)"
- "Util.sqlj -- Utilities for SQLJ sample programs (SQLj)"

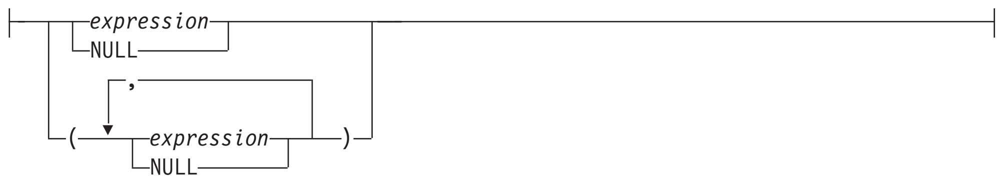
Fullselect



values-clause:



values-row:



The *fullselect* is a component of the select-statement, the INSERT statement, and the CREATE VIEW statement. It is also a component of certain predicates which, in turn, are components of a statement. A fullselect that is a component of a predicate is called a *subquery*, and a fullselect that is enclosed in parentheses is sometimes called a subquery.

The set operators UNION, EXCEPT, and INTERSECT correspond to the relational operators union, difference, and intersection.

A fullselect specifies a result table. If a set operator is not used, the result of the fullselect is the result of the specified subselect or values-clause.

values-clause
Derives a result table by specifying the actual values, using expressions, for each column of a row in the result table. Multiple rows may be specified.

NULL can only be used with multiple specifications of *values-row*, and at least one row in the same column must not be NULL (SQLSTATE 42826).

A *values-row* is specified by:

- A single expression for a single column result table or,
- *n* expressions (or NULL) separated by commas and enclosed in parentheses, where *n* is the number of columns in the result table.

A multiple row VALUES clause must have the same number of expressions in each *values-row* (SQLSTATE 42826).

The following are examples of values-clauses and their meaning.

```
VALUES (1),(2),(3)          - 3 rows of 1 column
VALUES 1, 2, 3              - 3 rows of 1 column
VALUES (1, 2, 3)            - 1 row of 3 columns
VALUES (1,21),(2,22),(3,23) - 3 rows of 2 columns
```

A values-clause that is composed of *n* specifications of *values-row*, RE_1 to RE_n, where *n* is greater than 1, is equivalent to:

RE_1 UNION ALL RE_2 ... UNION ALL RE_n

This means that the corresponding expressions of each *values-row* must be comparable (SQLSTATE 42825).

UNION or **UNION ALL**
Derives a result table by combining two other result tables (R1 and R2). If UNION ALL is specified, the result consists of all rows in R1 and R2. If UNION is specified without the ALL option, the result is the set of all rows in either R1 or R2, with the duplicate rows eliminated. In either case, however, each row of the UNION table is either a row from R1 or a row from R2.

EXCEPT or **EXCEPT ALL**
Derives a result table by combining two other result tables (R1 and R2). If EXCEPT ALL is specified, the result consists of all rows that do not have a corresponding row in R2, where duplicate rows are significant. If EXCEPT is specified without the ALL option, the result consists of all rows that are only in R1, with duplicate rows in the result of this operation eliminated.

INTERSECT or **INTERSECT ALL**
Derives a result table by combining two other result tables (R1 and R2). If INTERSECT ALL is specified, the result consists of all rows that are in both R1 and R2. If INTERSECT is specified without the ALL option, the result consists of all rows that are in both R1 and R2, with the duplicate rows eliminated.

order-by-clause

A fullselect that contains an ORDER BY or FETCH FIRST clause cannot be specified in:

- A materialized query table
- The outermost fullselect of a view (SQLSTATE 428FJ).

Note: An ORDER BY clause in a fullselect does not affect the order of the rows returned by a query. An ORDER BY clause only affects the order of the rows returned if it is specified in the outermost fullselect.

The number of columns in the result tables R1 and R2 must be the same (SQLSTATE 42826). If the ALL keyword is not specified, R1 and R2 must not include any columns having a data type of LONG VARCHAR, CLOB, LONG VARGRAPHIC, DBCLOB, BLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

The columns of the result are named as follows:

- If the *n*th column of R1 and the *n*th column of R2 have the same result column name, then the *n*th column of R has the result column name.
- If the *n*th column of R1 and the *n*th column of R2 have different result column names, a name is generated. This name cannot be used as the column name in an ORDER BY or UPDATE clause.

The generated name can be determined by performing a DESCRIBE of the SQL statement and consulting the SQLNAME field.

Two rows are duplicates of one another if each value in the first is equal to the corresponding value of the second. (For determining duplicates, two null values are considered equal.)

When multiple operations are combined in an expression, operations within parentheses are performed first. If there are no parentheses, the operations are performed from left to right with the exception that all INTERSECT operations are performed before UNION or EXCEPT operations.

In the following example, the values of tables R1 and R2 are shown on the left. The other headings listed show the values as a result of various set operations on R1 and R2.

R1	R2	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTER-SECT ALL	INTER-SECT
1	1	1	1	1	2	1	1
1	1	1	2	2	5	1	3
1	3	1	3	2		3	4
2	3	1	4	2		4	
2	3	1	5	4			
2	3	2		5			
3	4	2					
4		2					
4		3					
5		3					

R1	R2	UNION ALL	UNION	EXCEPT ALL	EXCEPT	INTER-SECT ALL	INTER-SECT
		3					
		3					
		3					
		4					
		4					
		4					
		5					

Examples of a fullselect

Example 1: Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

Example 2: List the employee numbers (EMPNO) of all employees in the EMPLOYEE table whose department number (WORKDEPT) either begins with 'E' **or** who are assigned to projects in the EMP_ACT table whose project number (PROJNO) equals 'MA2100', 'MA2110', or 'MA2112'.

```
SELECT EMPNO
   FROM EMPLOYEE
   WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO
   FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

Example 3: Make the same query as in example 2, and, in addition, “tag” the rows from the EMPLOYEE table with 'emp' and the rows from the EMP_ACT table with 'emp_act'. Unlike the result from example 2, this query may return the same EMPNO more than once, identifying which table it came from by the associated “tag”.

```
SELECT EMPNO, 'emp'
   FROM EMPLOYEE
   WHERE WORKDEPT LIKE 'E%'
UNION
SELECT EMPNO, 'emp_act' FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

Example 4: Make the same query as in example 2, only use UNION ALL so that no duplicate rows are eliminated.

```
SELECT EMPNO
    FROM EMPLOYEE
    WHERE WORKDEPT LIKE 'E%'
UNION ALL
SELECT EMPNO
   FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
```

Example 5: Make the same query as in Example 3, only include an additional two employees currently not in any table and tag these rows as ″new″.

```
SELECT EMPNO, 'emp'
   FROM EMPLOYEE
   WHEREWORKDEPTLIKE 'E%'
UNION
```

```
SELECT EMPNO, 'emp_act'
   FROM EMP_ACT
   WHERE PROJNO IN('MA2100', 'MA2110', 'MA2112')
UNION
   VALUES ('NEWAAA', 'new'), ('NEWBBB', 'new')
```

Example 6: This example of EXCEPT produces all rows that are in T1 but not in T2.

```
(SELECT * FROM T1)
EXCEPT ALL
(SELECT * FROM T2)
```

If no NULL values are involved, this example returns the same results as

```
SELECT ALL *
   FROM T1
   WHERE NOT EXISTS (SELECT * FROM T2
                       WHERE T1.C1 = T2.C1 AND T1.C2 = T2.C2 AND...)
```

Example 7: This example of INTERSECT produces all rows that are in both tables T1 and T2, removing duplicates.

```
(SELECT * FROM T1)
INTERSECT
(SELECT * FROM T2)
```

If no NULL values are involved, this example returns the same result as

```
SELECT DISTINCT * FROM T1
   WHERE EXISTS (SELECT * FROM T2
                 WHERE T1.C1 = T2.C1 AND T1.C2 = T2.C2 AND...)
```

where C1, C2, and so on represent the columns of T1 and T2.

Related reference:
- "Rules for result data types" in *SQL Reference, Volume 1*
- "Rules for string conversions" in *SQL Reference, Volume 1*

LOCK TABLE

The LOCK TABLE statement prevents concurrent application processes from using or changing a table.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- SELECT privilege on the table
- CONTROL privilege on the table
- SYSADM or DBADM authority

Syntax:

```
►►──LOCK TABLE──┬─table-name─┬──IN──┬─SHARE─────┬──MODE──────────────────────►◄
                └─nickname───┘      └─EXCLUSIVE─┘
```

Description:

table-name **or** *nickname*
: Identifies the table or nickname. The *table-name* must identify a table that exists at the application server, but it must not identify a catalog table, or a declared temporary table (SQLSTATE 42995). If the *table-name* is a typed table, it must be the root table of the table hierarchy (SQLSTATE 428DR). When a nickname is specified, DB2 will lock the underlying object (that is, a table or view) of the data source to which the nickname refers.

IN SHARE MODE
: Prevents concurrent application processes from executing any but read-only operations on the table.

IN EXCLUSIVE MODE
: Prevents concurrent application processes from executing any operations on the table. Note that EXCLUSIVE MODE does not prevent concurrent application processes that are running at isolation level Uncommitted Read (UR) from executing read-only operations on the table.

Notes:

- Locking is used to prevent concurrent operations. A lock is not necessarily acquired during execution of the LOCK TABLE statement if a suitable lock already exists. The lock that prevents concurrent operations is held at least until termination of the unit of work.
- In a partitioned database, a table lock is first acquired at the first database partition in the database partition group (the database partition with the lowest number) and then at other database partitions. If the LOCK TABLE statement is interrupted, the table may be locked on some database partitions but not on others. If this occurs, either issue another LOCK TABLE statement to complete the locking on all database partitions, or issue a COMMIT or ROLLBACK statement to release the current locks.
- This statement affects all database partitions in the database partition group.
- For partitioned tables, the only lock acquired for the LOCK TABLE statement is at the table level; no data partition locks are acquired.

Example:

Obtain a lock on the table EMP. Do not allow other programs to read or update the table.

```
   LOCK TABLE EMP IN EXCLUSIVE MODE
```

ROLLBACK

The ROLLBACK statement is used to back out of the database changes that were made within a unit of work or a savepoint.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared.

Authorization:

None required.

Syntax:

```
           ┌─WORK─┐
►►─ROLLBACK─┴──────┴──┬──────────────────────────────────┬──────────────────►◄
                      └─TO SAVEPOINT─┬────────────────┬─┘
                                     └─savepoint-name─┘
```

Description:
The unit of work in which the ROLLBACK statement is executed is terminated and a new unit of work is initiated. All changes made to the database during the unit of work are backed out.

The following statements, however, are not under transaction control, and changes made by them are independent of the ROLLBACK statement:
- SET CONNECTION
- SET CURRENT DEFAULT TRANSFORM GROUP
- SET CURRENT DEGREE
- SET CURRENT EXPLAIN MODE
- SET CURRENT EXPLAIN SNAPSHOT
- SET CURRENT LOCK TIMEOUT
- SET CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION
- SET CURRENT PACKAGESET
- SET CURRENT QUERY OPTIMIZATION
- SET CURRENT REFRESH AGE
- SET ENCRYPTION PASSWORD
- SET EVENT MONITOR STATE
- SET PASSTHRU

  Note: Although the SET PASSTHRU statement is not under transaction control, the passthru session initiated by the statement is under transaction control.
- SET PATH
- SET SCHEMA
- SET SERVER OPTION

The generation of sequence and identity values is not under transaction control. Values generated and consumed by the *nextval-expression* or by inserting rows into a table that has an identity column are independent of issuing the ROLLBACK statement. Also, issuing the ROLLBACK statement does not affect the value returned by the *prevval-expression*, nor the IDENTITY_VAL_LOCAL function.

TO SAVEPOINT
Specifies that a partial rollback (ROLLBACK TO SAVEPOINT) is to be performed. If no savepoint is active in the current savepoint level (see the "Rules" section in the description of the SAVEPOINT statement), an error is returned (SQLSTATE 3B502). After a successful rollback, the savepoint continues to exist, but any nested savepoints are released and no longer exist. The nested savepoints, if any, are considered to have been rolled back and then

released as part of the rollback to the current savepoint. If a *savepoint-name* is not provided, rollback occurs to the most recently set savepoint within the current savepoint level.

If this clause is omitted, the ROLLBACK statement rolls back the entire transaction. Furthermore, savepoints within the transaction are released.

savepoint-name
Specifies the savepoint that is to be used in the rollback operation. The specified *savepoint-name* cannot begin with ’SYS’ (SQLSTATE 42939). After a successful rollback operation, the named savepoint continues to exist. If the savepoint name does not exist, an error (SQLSTATE 3B001) is returned. Data and schema changes made since the savepoint was set are undone.

Notes:

- All locks held are released on a ROLLBACK of the unit of work. All open cursors are closed. All LOB locators are freed.
- Executing a ROLLBACK statement does not affect either the SET statements that change special register values or the RELEASE statement.
- If the program terminates abnormally, the unit of work is implicitly rolled back.
- Statement caching is affected by the rollback operation.
- The impact on cursors resulting from a ROLLBACK TO SAVEPOINT depends on the statements within the savepoint
  - If the savepoint contains DDL on which a cursor is dependent, the cursor is marked invalid. Attempts to use such a cursor results in an error (SQLSTATE 57007).
  - Otherwise:
    - If the cursor is referenced in the savepoint, the cursor remains open and is positioned before the next logical row of the result table. (A FETCH must be performed before a positioned UPDATE or DELETE statement is issued.)
    - Otherwise, the cursor is not affected by the ROLLBACK TO SAVEPOINT (it remains open and positioned).
- Dynamically prepared statement names are still valid, although the statement may be implicitly prepared again, as a result of DDL operations that are rolled back within the savepoint.
- A ROLLBACK TO SAVEPOINT operation will drop any declared temporary tables named within the savepoint. If a declared temporary table is modified within the savepoint, then all rows in the table are deleted.
- All locks are retained after a ROLLBACK TO SAVEPOINT statement.
- All LOB locators are preserved following a ROLLBACK TO SAVEPOINT operation.

Example:

Delete the alterations made since the last commit point or rollback.

```
ROLLBACK WORK
```

Related reference:

- “EXECUTE statement” in *SQL Reference, Volume 2*
- “SAVEPOINT statement” in *SQL Reference, Volume 2*

Related samples:

- “delet.sqb -- How to delete table data (MF COBOL)”

- "spclient.sqc -- Call various stored procedures (C)"
- "spclient.sqC -- Call various stored procedures (C++)"

SELECT

The SELECT statement is a form of query. It can be embedded in an application program or issued interactively.

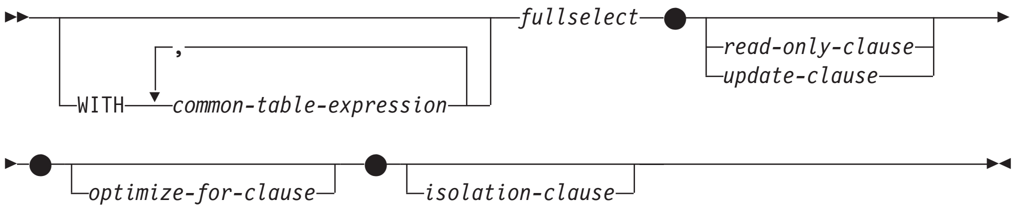
Related reference:

- "Select-statement" on page 1182
- "Subselect" on page 1211

Related samples:

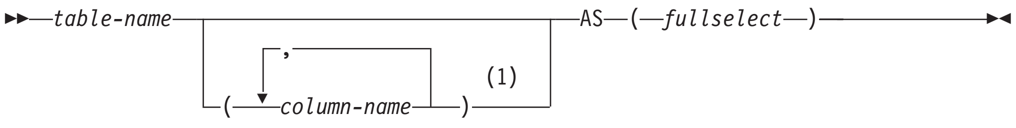
- "dynamic.sqb -- How to update table data with cursor dynamically (MF COBOL)"
- "static.sqb -- Get table data using static SQL statement (MF COBOL)"
- "tbread.c -- How to read data from tables"
- "tbread.sqc -- How to read tables (C)"
- "tbread.sqC -- How to read tables (C++)"
- "TbRead.java -- How to read table data (JDBC)"
- "TbRead.sqlj -- How to read table data (SQLj)"

Select-statement



The *select-statement* is the form of a query that can be directly specified in a DECLARE CURSOR statement, or prepared and then referenced in a DECLARE CURSOR statement. It can also be issued through the use of dynamic SQL statements using the command line processor (or similar tools), causing a result table to be displayed on the user's screen. In either case, the table specified by a *select-statement* is the result of the fullselect.

common-table-expression



Notes:

1 If a common table expression is recursive, or if the fullselect results in duplicate column names, column names must be specified.

A *common table expression* permits defining a result table with a *table-name* that can be specified as a table name in any FROM clause of the fullselect that follows. Multiple common table expressions can be specified following the single WITH keyword. Each common table expression specified can also be referenced by name in the FROM clause of subsequent common table expressions.

If a list of columns is specified, it must consist of as many names as there are columns in the result table of the fullselect. Each *column-name* must be unique and unqualified. If these column names are not specified, the names are derived from the select list of the fullselect used to define the common table expression.

The *table-name* of a common table expression must be different from any other common table expression *table-name* in the same statement (SQLSTATE 42726). If the common table expression is specified in an INSERT statement the *table-name* cannot be the same as the table or view name that is the object of the insert (SQLSTATE 42726). A common table expression *table-name* can be specified as a table name in any FROM clause throughout the fullselect. A *table-name* of a common table expression overrides any existing table, view or alias (in the catalog) with the same qualified name.

If more than one common table expression is defined in the same statement, cyclic references between the common table expressions are not permitted (SQLSTATE 42835). A *cyclic reference* occurs when two common table expressions *dt1* and *dt2* are created such that *dt1* refers to *dt2* and *dt2* refers to *dt1*.

If the fullselect of a common table expression contains a *data-change-table-reference* in the FROM clause, the common table expression is said to modify data. A common table expression that modifies data is always evaluated when the statement is processed, regardless of whether the common table expression is used anywhere else in the statement. If there is at least one common table expression that reads or modifies data, all common table expressions are processed in the order in which they occur, and each common table expression that reads or modifies data is completely executed, including all constraints and triggers, before any subsequent common table expressions are executed.

The common table expression is also optional prior to the fullselect in the CREATE VIEW and INSERT statements.

A common table expression can be used:
- In place of a view to avoid creating the view (when general use of the view is not required and positioned updates or deletes are not used)
- To enable grouping by a column that is derived from a scalar subselect or function that is not deterministic or has external action
- When the desired result table is based on host variables
- When the same result table needs to be shared in a fullselect
- When the result needs to be derived using recursion
- When multiple SQL data change statements need to be processed within the query

If the fullselect of a common table expression contains a reference to itself in a FROM clause, the common table expression is a *recursive common table expression*. Queries using recursion are useful in supporting applications such as bill of materials (BOM), reservation systems, and network planning.

The following must be true of a recursive common table expression:

- Each fullselect that is part of the recursion cycle must start with SELECT or SELECT ALL. Use of SELECT DISTINCT is not allowed (SQLSTATE 42925). Furthermore, the unions must use UNION ALL (SQLSTATE 42925).
- The column names must be specified following the *table-name* of the common table expression (SQLSTATE 42908).
- The first fullselect of the first union (the initialization fullselect) must not include a reference to any column of the common table expression in any FROM clause (SQLSTATE 42836).
- If a column name of the common table expression is referred to in the iterative fullselect, the data type, length, and code page for the column are determined based on the initialization fullselect. The corresponding column in the iterative fullselect must have the same data type and length as the data type and length determined based on the initialization fullselect and the code page must match (SQLSTATE 42825). However, for character string types, the length of the two data types may differ. In this case, the column in the iterative fullselect must have a length that would always be assignable to the length determined from the initialization fullselect.
- Each fullselect that is part of the recursion cycle must not include any column functions, group-by-clauses, or having-clauses (SQLSTATE 42836).

  The FROM clauses of these fullselects can include at most one reference to a common table expression that is part of a recursion cycle (SQLSTATE 42836).
- The iterative fullselect and the overall recursive fullselect must not include an order-by-clause (SQLSTATE 42836).
- Subqueries (scalar or quantified) must not be part of any recursion cycles (SQLSTATE 42836).

When developing recursive common table expressions, remember that an infinite recursion cycle (loop) can be created. Check that recursion cycles will terminate. This is especially important if the data involved is cyclic. A recursive common table expression is expected to include a predicate that will prevent an infinite loop. The recursive common table expression is expected to include:

- In the iterative fullselect, an integer column incremented by a constant.
- A predicate in the where clause of the iterative fullselect in the form ″counter_col < constant″ or ″counter _col < :hostvar″.

A warning is issued if this syntax is not found in the recursive common table expression (SQLSTATE 01605).

Recursion example: bill of materials:

Bill of materials (BOM) applications are a common requirement in many business environments. To illustrate the capability of a recursive common table expression for BOM applications, consider a table of parts with associated subparts and the quantity of subparts required by the part. For this example, create the table as follows:

```
CREATE TABLE PARTLIST
              (PART VARCHAR(8),
               SUBPART VARCHAR(8),
               QUANTITY INTEGER);
```

To give query results for this example, assume that the PARTLIST table is populated with the following values:

```
PART     SUBPART  QUANTITY
-------- -------- -----------
00       01                 5
00       05                 3
01       02                 2
01       03                 3
01       04                 4
01       06                 3
02       05                 7
02       06                 6
03       07                 6
04       08                10
04       09                11
05       10                10
05       11                10
06       12                10
06       13                10
07       14                 8
07       12                 8
```

Example 1: Single level explosion

The first example is called single level explosion. It answers the question, "What parts are needed to build the part identified by '01'?". The list will include the direct subparts, subparts of the subparts and so on. However, if a part is used multiple times, its subparts are only listed once.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
     (  SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
        FROM PARTLIST ROOT
        WHERE ROOT.PART = '01'
      UNION ALL
        SELECT CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
        FROM RPL PARENT, PARTLIST CHILD
        WHERE  PARENT.SUBPART = CHILD.PART
     )
SELECT DISTINCT PART, SUBPART, QUANTITY
 FROM RPL
  ORDER BY PART, SUBPART, QUANTITY;
```

The above query includes a common table expression, identified by the name *RPL*, that expresses the recursive part of this query. It illustrates the basic elements of a recursive common table expression.

The first operand (fullselect) of the UNION, referred to as the *initialization fullselect*, gets the direct children of part '01'. The FROM clause of this fullselect refers to the source table and will never refer to itself (*RPL* in this case). The result of this first fullselect goes into the common table expression *RPL* (Recursive PARTLIST). As in this example, the UNION must always be a UNION ALL.

The second operand (fullselect) of the UNION uses *RPL* to compute subparts of subparts by having the FROM clause refer to the common table expression *RPL* and the source table with a join of a part from the source table (child) to a subpart of the current result contained in *RPL* (parent). The result goes back to *RPL* again. The second operand of UNION is then used repeatedly until no more children exist.

The SELECT DISTINCT in the main fullselect of this query ensures the same part/subpart is not listed more than once.

The result of the query is as follows:

```
PART     SUBPART  QUANTITY
-------- -------- -----------
01       02                 2
01       03                 3
01       04                 4
01       06                 3
02       05                 7
02       06                 6
03       07                 6
04       08                10
04       09                11
05       10                10
05       11                10
06       12                10
06       13                10
07       12                 8
07       14                 8
```

Observe in the result that from part ’01’ we go to ’02’ which goes to ’06’ and so on. Further, notice that part ’06’ is reached twice, once through ’01’ directly and another time through ’02’. In the output, however, its subcomponents are listed only once (this is the result of using a SELECT DISTINCT) as required.

It is important to remember that with recursive common table expressions it is possible to introduce an *infinite loop*. In this example, an infinite loop would be created if the search condition of the second operand that joins the parent and child tables was coded as:

```
PARENT.SUBPART = CHILD.SUBPART
```

This example of causing an infinite loop is obviously a case of not coding what is intended. However, care should also be exercised in determining what to code so that there is a definite end of the recursion cycle.

The result produced by this example query could be produced in an application program without using a recursive common table expression. However, this approach would require starting of a new query for every level of recursion. Furthermore, the application needs to put all the results back in the database to order the result. This approach complicates the application logic and does not perform well. The application logic becomes even harder and more inefficient for other bill of material queries, such as summarized and indented explosion queries.

Example 2: Summarized explosion

The second example is a summarized explosion. The question posed here is, what is the total quantity of each part required to build part ’01’. The main difference from the single level explosion is the need to aggregate the quantities. The first example indicates the quantity of subparts required for the part whenever it is required. It does not indicate how many of the subparts are needed to build part ’01’.

```
WITH RPL (PART, SUBPART, QUANTITY) AS
   (
      SELECT ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
       FROM PARTLIST ROOT
       WHERE ROOT.PART = '01'
    UNION ALL
      SELECT PARENT.PART, CHILD.SUBPART, PARENT.QUANTITY*CHILD.QUANTITY
       FROM RPL PARENT, PARTLIST CHILD
       WHERE PARENT.SUBPART = CHILD.PART
   )
```

```
SELECT PART, SUBPART, SUM(QUANTITY) AS "Total QTY Used"
 FROM RPL
  GROUP BY PART, SUBPART
  ORDER BY PART, SUBPART;
```

In the above query, the select list of the second operand of the UNION in the recursive common table expression, identified by the name *RPL*, shows the aggregation of the quantity. To find out how much of a subpart is used, the quantity of the parent is multiplied by the quantity per parent of a child. If a part is used multiple times in different places, it requires another final aggregation. This is done by the grouping over the common table expression *RPL* and using the SUM column function in the select list of the main fullselect.

The result of the query is as follows:

```
  PART     SUBPART  Total Qty Used
  -------- -------- --------------
  01       02                    2
  01       03                    3
  01       04                    4
  01       05                   14
  01       06                   15
  01       07                   18
  01       08                   40
  01       09                   44
  01       10                  140
  01       11                  140
  01       12                  294
  01       13                  150
  01       14                  144
```

Looking at the output, consider the line for subpart ’06’. The total quantity used value of 15 is derived from a quantity of 3 directly for part ’01’ and a quantity of 6 for part ’02’ which is needed 2 times by part ’01’.

Example 3: Controlling depth

The question may come to mind, what happens when there are more levels of parts in the table than you are interested in for your query? That is, how is a query written to answer the question, “What are the first two levels of parts needed to build the part identified by ’01’?” For the sake of clarity in the example, the level is included in the result.

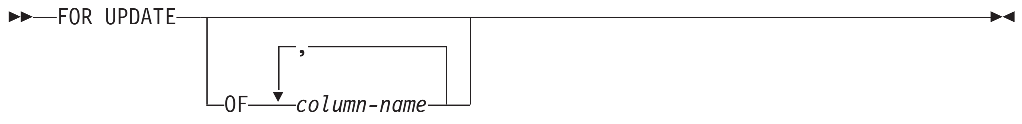
```
WITH RPL (LEVEL, PART, SUBPART, QUANTITY) AS
      (
         SELECT 1,              ROOT.PART, ROOT.SUBPART, ROOT.QUANTITY
          FROM PARTLIST ROOT
          WHERE ROOT.PART = '01'
       UNION ALL
         SELECT PARENT.LEVEL+1, CHILD.PART, CHILD.SUBPART, CHILD.QUANTITY
          FROM RPL PARENT, PARTLIST CHILD
          WHERE PARENT.SUBPART = CHILD.PART
            AND PARENT.LEVEL < 2
      )
 SELECT PART, LEVEL, SUBPART, QUANTITY
   FROM RPL;
```

This query is similar to example 1. The column *LEVEL* was introduced to count the levels from the original part. In the initialization fullselect, the value for the *LEVEL* column is initialized to 1. In the subsequent fullselect, the level from the parent is incremented by 1. Then to control the number of levels in the result, the second fullselect includes the condition that the parent level must be less than 2. This ensures that the second fullselect only processes children to the second level.

The result of the query is:

```
PART     LEVEL       SUBPART  QUANTITY
-------- ----------- -------- -----------
01                 1 02                 2
01                 1 03                 3
01                 1 04                 4
01                 1 06                 3
02                 2 05                 7
02                 2 06                 6
03                 2 07                 6
04                 2 08                10
04                 2 09                11
06                 2 12                10
06                 2 13                10
```

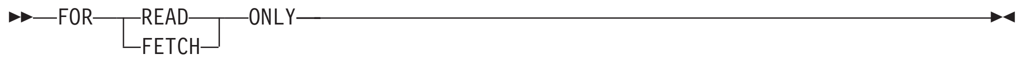
update-clause



The FOR UPDATE clause identifies the columns that can be updated in a subsequent Positioned UPDATE statement. Each *column-name* must be unqualified and must identify a column of the table or view identified in the first FROM clause of the fullselect. If the FOR UPDATE clause is specified without column names, all updatable columns of the table or view identified in the first FROM clause of the fullselect are included.

The FOR UPDATE clause cannot be used if one of the following is true:
- The cursor associated with the select-statement is not deletable .
- One of the selected columns is a non-updatable column of a catalog table and the FOR UPDATE clause has not been used to exclude that column.

read-only-clause



The FOR READ ONLY clause indicates that the result table is read-only and therefore the cursor cannot be referred to in Positioned UPDATE and DELETE statements. FOR FETCH ONLY has the same meaning.

Some result tables are read-only by nature. (For example, a table based on a read-only view.) FOR READ ONLY can still be specified for such tables, but the specification has no effect.

For result tables in which updates and deletes are allowed, specifying FOR READ ONLY (or FOR FETCH ONLY) can possibly improve the performance of FETCH operations by allowing the database manager to do blocking. For example, in programs that contain dynamic SQL statements without the FOR READ ONLY or ORDER BY clause, the database manager might open cursors as if the FOR UPDATE clause were specified. It is recommended, therefore, that the FOR READ ONLY clause be used to improve performance, except in cases where queries will be used in positioned UPDATE or DELETE statements.

A read-only result table must not be referred to in a Positioned UPDATE or DELETE statement, whether it is read-only by nature or specified as FOR READ ONLY (FOR FETCH ONLY).

optimize-for-clause

```
►►──OPTIMIZE FOR──integer──┬─ROWS─┬──────────────────────────────────────►◄
                           └─ROW──┘
```

The OPTIMIZE FOR clause requests special processing of the *select statement*. If the clause is omitted, it is assumed that all rows of the result table will be retrieved; if it is specified, it is assumed that the number of rows retrieved will probably not exceed *n*, where *n* is the value of *integer*. The value of *n* must be a positive integer. Use of the OPTIMIZE FOR clause influences query optimization, based on the assumption that *n* rows will be retrieved. In addition, for cursors that are blocked, this clause will influence the number of rows that will be returned in each block (that is, no more than *n* rows will be returned in each block). If both the *fetch-first-clause* and the *optimize-for-clause* are specified, the lower of the integer values from these clauses will be used to influence the communications buffer size. The values are considered independently for optimization purposes.

This clause does not limit the number of rows that can be fetched, or affect the result in any other way than performance. Using OPTIMIZE FOR *n* ROWS can improve performance if no more than *n* rows are retrieved, but may degrade performance if more than *n* rows are retrieved.

If the value of *n* multiplied by the size of the row exceeds the size of the communication buffer, the OPTIMIZE FOR clause will have no impact on the data buffers. The size of the communication buffer is defined by the RQRIOBLK or the ASLHEAPSZ configuration parameter.

isolation-clause

```
►►──┬─────────────────────────────────────────────┬──────────────────────►◄
    └─WITH──┬─RR──┬────────────────────────┬─┬─┘
            │     └─lock-request-clause─┘ │
            ├─RS──┬────────────────────────┬─┤
            │     └─lock-request-clause─┘ │
            ├─CS─────────────────────────────┤
            └─UR─────────────────────────────┘
```

The optional *isolation-clause* specifies the isolation level at which the statement is executed, and whether a specific type of lock is to be acquired.

- RR - Repeatable Read
- RS - Read Stability
- CS - Cursor Stability
- UR - Uncommitted Read

The default isolation level of the statement is the isolation level of the package in which the statement is bound. When a nickname is used in a *select-statement* to access data in DB2 family and Microsoft SQL Server data sources, the *isolation-clause* can be included in the statement to specify the statement isolation level. If the *isolation-clause* is included in statements that access other data sources, the specified isolation level is ignored. The current isolation level on the federated server is mapped to a corresponding isolation level at the data source on each connection to the data source. After a connection is made to a data source, the isolation level cannot be changed for the duration of the connection.

lock-request-clause

```
►►──USE AND KEEP──┬─SHARE─────┬──LOCKS─────────────────────────────────────►◄
                  ├─UPDATE────┤
                  └─EXCLUSIVE─┘
```

The optional *lock-request-clause* specifies the type of lock that the database manager is to acquire and hold:

SHARE	Concurrent processes can acquire SHARE or UPDATE locks on the data.
UPDATE	Concurrent processes can acquire SHARE locks on the data, but no concurrent process can acquire an UPDATE or EXCLUSIVE lock.
EXCLUSIVE	Concurrent processes cannot acquire a lock on the data.

The *lock-request-clause* applies to all base table and index scans required by the query, including those within subqueries, SQL functions and SQL methods. It has no affect on locks placed by procedures, external functions, or external methods. Any SQL function or SQL method invoked (directly or indirectly) by the statement must be created with INHERIT ISOLATION LEVEL WITH LOCK REQUEST (SQLSTATE 42601). The *lock-request-clause* cannot be used with a modifying query that might invoke triggers or that requires referential integrity checks (SQLSTATE 42601).

Examples of a select-statement

Example 1: Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

Example 2: Select the project name (PROJNAME), start date (PRSTDATE), and end date (PRENDATE) from the PROJECT table. Order the result table by the end date with the most recent dates appearing first.

```
SELECT PROJNAME, PRSTDATE, PRENDATE
 FROM PROJECT
 ORDER BY PRENDATE DESC
```

Example 3: Select the department number (WORKDEPT) and average departmental salary (SALARY) for all departments in the EMPLOYEE table. Arrange the result table in ascending order by average departmental salary.

```
SELECT WORKDEPT, AVG(SALARY)
   FROM EMPLOYEE
   GROUP BY WORKDEPT
   ORDER BY 2
```

Example 4: Declare a cursor named UP_CUR to be used in a C program to update the start date (PRSTDATE) and the end date (PRENDATE) columns in the PROJECT table. The program must receive both of these values together with the project number (PROJNO) value for each row.

```
EXEC SQL  DECLARE UP_CUR CURSOR FOR
            SELECT PROJNO, PRSTDATE, PRENDATE
              FROM PROJECT
              FOR UPDATE OF PRSTDATE, PRENDATE;
```

Example 5: This example names the expression SAL+BONUS+COMM as TOTAL_PAY

```
SELECT SALARY+BONUS+COMM AS TOTAL_PAY
  FROM EMPLOYEE
  ORDER BY TOTAL_PAY
```

Example 6: Determine the employee number and salary of sales representatives along with the average salary and head count of their departments. Also, list the average salary of the department with the highest average salary.

Using a common table expression for this case saves the overhead of creating the DINFO view as a regular view. During statement preparation, accessing the catalog for the view is avoided and, because of the context of the rest of the fullselect, only the rows for the department of the sales representatives need to be considered by the view.

```
WITH
   DINFO (DEPTNO, AVGSALARY, EMPCOUNT) AS
       (SELECT OTHERS.WORKDEPT, AVG(OTHERS.SALARY), COUNT(*)
         FROM EMPLOYEE OTHERS
         GROUP BY OTHERS.WORKDEPT
       ),
   DINFOMAX AS
       (SELECT MAX(AVGSALARY) AS AVGMAX FROM DINFO)
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY,
       DINFO.AVGSALARY, DINFO.EMPCOUNT, DINFOMAX.AVGMAX
 FROM EMPLOYEE THIS_EMP, DINFO, DINFOMAX
 WHERE THIS_EMP.JOB = 'SALESREP'
 AND THIS_EMP.WORKDEPT = DINFO.DEPTNO
```

Example 7: Given two tables, EMPLOYEE and PROJECT, replace employee SALLY with a new employee GEORGE, assign all projects lead by SALLY to GEORGE, and return the names of the updated projects.

```
WITH
  NEWEMP AS (SELECT EMPNO FROM NEW TABLE
             (INSERT INTO EMPLOYEE(EMPNO, FIRSTNME)
                 VALUES(NEXT VALUE FOR EMPNO_SEQ, 'GEORGE'))),
  OLDEMP AS (SELECT EMPNO FROM EMPLOYEE WHERE FIRSTNME = 'SALLY'),
  UPPROJ AS (SELECT PROJNAME FROM NEW TABLE
             (UPDATE PROJECT
                 SET RESPEMP = (SELECT EMPNO FROM NEWEMP)
                 WHERE RESPEMP = (SELECT EMPNO FROM OLDEMP))),
  DELEMP AS (SELECT EMPNO FROM OLD TABLE
             (DELETE FROM EMPLOYEE
                 WHERE EMPNO = (SELECT EMPNO FROM OLDEMP)))
SELECT PROJNAME FROM UPPROJ;
```

Example 8: Retrieve data from the DEPT table. That data will later be updated with a searched update, and should be locked when the query executes.

```
SELECT DEPTNO, DEPTNAME, MGRNO
  FROM DEPT
  WHERE ADMRDEPT ='A00'
  FOR READ ONLY WITH RS USE AND KEEP EXCLUSIVE LOCKS
```

Related reference:

- "DECLARE CURSOR statement" in *SQL Reference, Volume 2*

- "Subselect" on page 1211

SET INTEGRITY

The SET INTEGRITY statement is used to:

- Bring one or more tables out of set integrity pending state (previously known as ″check pending state″) by performing required integrity processing on those tables.
- Bring one or more tables out of set integrity pending state without performing required integrity processing on those tables.
- Place one or more tables in set integrity pending state.
- Place one or more tables into full access state.
- Prune the contents of one or more staging tables.

When the statement is used to perform integrity processing for a table after it has been loaded or attached, the system can incrementally process the table by checking only the appended portion for constraints violations. If the subject table is a materialized query table or a staging table, and load, attach, or detach operations are performed on its underlying tables, the system can incrementally refresh the materialized query table or incrementally propagate to the staging table with only the delta portions of its underlying tables. However, there are some situations in which the system will not be able to perform such optimizations and will instead perform full integrity processing to ensure data integrity. Full integrity processing is done by checking the entire table for constraints violations, recomputing a materialized query table's definition, or marking a staging table as inconsistent. The latter implies that a full refresh of its associated materialized query table is required. There is also a situation in which you might want to explicitly request incremental processing by specifying the INCREMENTAL option.

The SET INTEGRITY statement is under transaction control.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges required to execute the SET INTEGRITY statement depend on the purpose, as outlined below.

- Bringing tables out of set integrity pending state and performing the required integrity processing.

  The privileges held by the authorization ID of the statement must include at least one of the following:

  – CONTROL privilege on:

    - The tables on which integrity processing is performed and, if exception tables are provided for one or more of those tables, INSERT privilege on the exception tables
    - All descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables that will implicitly be placed in set integrity pending state by the statement

  - LOAD authority (with conditions). The following conditions must all be met before LOAD authority can be considered as providing valid privileges:
    - The required integrity processing does not involve the following actions:
      - Refreshing a materialized query table
      - Propagating to a staging table
      - Updating a generated or identity column
    - If exception tables are provided for one or more tables, the required access is granted for the duration of the integrity processing to the tables on which integrity processing is performed, and to the associated exception tables. That is:
      - SELECT and DELETE privilege on each table on which integrity processing is performed, and
      - INSERT privilege on the exception tables
  - SYSADM or DBADM authority
- Bringing tables out of set integrity pending state without performing the required integrity processing.

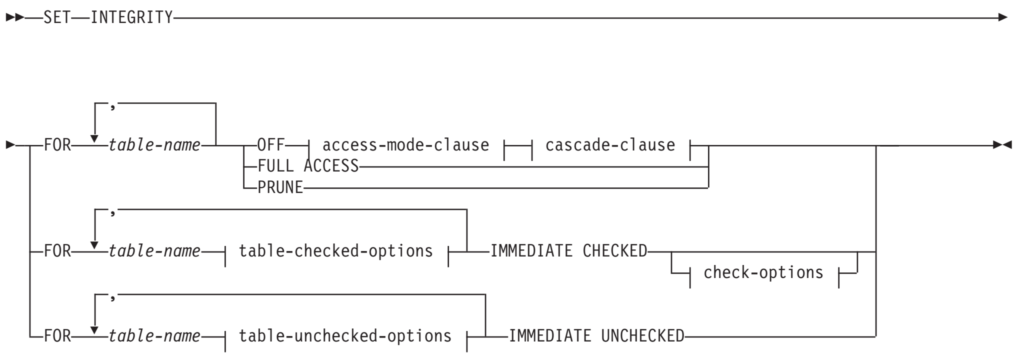
  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on the tables that are being processed; CONTROL privilege on each descendent foreign key table, descendent immediate materialized query table, and descendent immediate staging table that will implicitly be placed in set integrity pending state by the statement
  - LOAD authority
  - SYSADM or DBADM authority
- Placing tables in set integrity pending state.

  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on:
    - The specified tables, and
    - The descendent foreign key tables that will be placed in set integrity pending state by the statement, and
    - The descendent immediate materialized query tables that will be placed in set integrity pending state by the statement, and
    - The descendent immediate staging tables that will be placed in set integrity pending state by the statement
  - LOAD authority
  - SYSADM or DBADM authority
- Place a table into the full access state.

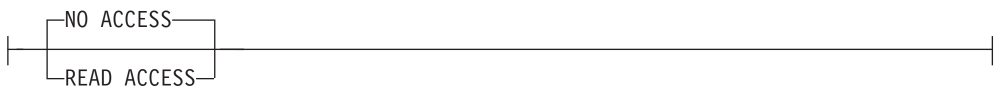
  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on the tables that are placed into the full access state
  - LOAD authority
  - SYSADM or DBADM authority
- Prune a staging table.

  The privileges held by the authorization ID of the statement must include at least one of the following:
  - CONTROL privilege on the table being pruned
  - SYSADM or DBADM authority

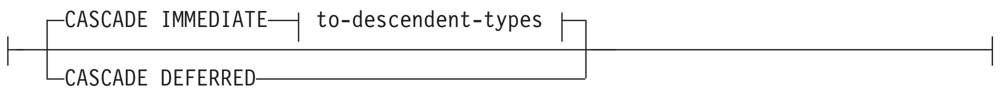
Syntax:



access-mode-clause:



cascade-clause:



to-descendent-types:

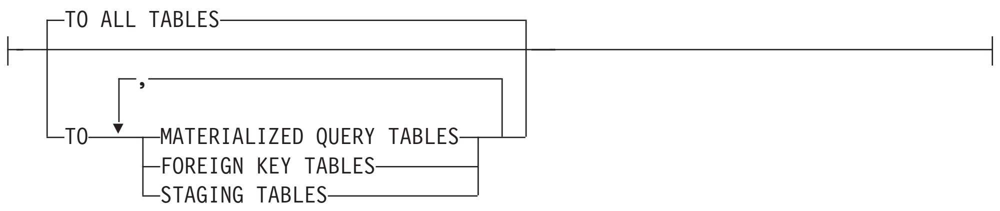
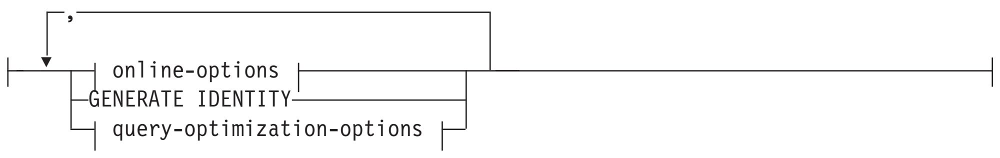
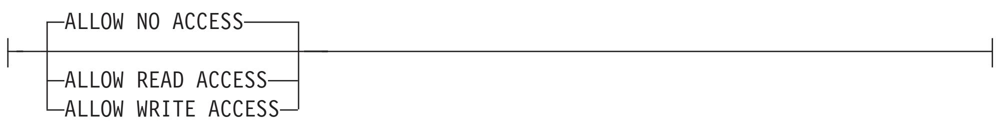


table-checked-options:



online-options:



query-optimization-options:

```
|--+-----------------------------------------------------------------------------------------------+--|
   '-+-------------------------+--USING REFRESH DEFERRED TABLES--+-----------------------+-'
     '-ALLOW QUERY OPTIMIZATION-'                                 '-WITH REFRESH AGE ANY-'
```

check-options:

```
|--●--| incremental-options |--●--+-----------------+--●--+-------+------------->
                                 '-FORCE GENERATED-'     '-PRUNE-'

>--●--+-------------+--●--+--------------------------+-------------------------|
      '-FULL ACCESS-'     '-| exception-clause |-'
```

incremental-options:

```
|--+-----------------+-------------------------------------------------------|
   +-INCREMENTAL-----+
   '-NOT INCREMENTAL-'
```

exception-clause:

```
                  .-,--------------------------.
                  V                            |
|--FOR EXCEPTION----| in-table-use-clause |--+-------------------------------|
```

in-table-use-clause:

```
|--IN--table-name--USE--table-name-------------------------------------------|
```

table-unchecked-options:

```
   .-,------------------------------------------------.
   V                                                  |
|----| integrity-options |--+-------------+--+----------------------------------|
                              '-FULL ACCESS-'
```

integrity-options:

```
|--+-ALL---------------------------+-----------------------------------------|
   |   .-,-------------------------.   |
   |   V                           |   |
   '-----+-FOREIGN KEY---------+--+--'
         +-CHECK---------------+
         +-MATERIALIZED QUERY--+
         +-GENERATED COLUMN----+
         '-STAGING-------------'
```

Description:

FOR *table-name*
: Identifies one or more tables for integrity processing. It must be a table described in the catalog and must not be a view, catalog table, or typed table.

OFF
Specifies that the tables are placed in set integrity pending state. Only very limited activity is allowed on a table that is in set integrity pending state.

access-mode-clause
Specifies the readability of the table while it is in set integrity pending state.

NO ACCESS
Specifies that the table is to be put in set integrity pending no access state, which does not allow read or write access to the table.

READ ACCESS
Specifies that the table is to be put in set integrity pending read access state, which allows read access to the non-appended portion of the table. This option is not allowed on a table that is in set integrity pending no access state (SQLSTATE 428FH).

cascade-clause
Specifies whether the set integrity pending state of the table referenced in the SET INTEGRITY statement is to be immediately cascaded to descendent tables.

CASCADE IMMEDIATE
Specifies that the set integrity pending state is to be immediately extended to descendent tables.

to-descendent-types
Specifies the type of descendent tables to which the set integrity pending state is immediately cascaded.

TO ALL TABLES
Specifies that the set integrity pending state is to be immediately cascaded to all descendent tables of the tables in the invocation list. Descendent tables include all descendent foreign key tables, immediate staging tables, and immediate materialized query tables that are descendants of the tables in the invocation list, or descendants of descendent foreign key tables.

Specifying TO ALL TABLES is equivalent to specifying TO FOREIGN KEY TABLES, TO MATERIALIZED QUERY TABLES, and TO STAGING TABLES, all in the same statement.

TO MATERIALIZED QUERY TABLES
If only TO MATERIALIZED QUERY TABLES is specified, the set integrity pending state is to be immediately cascaded only to descendent immediate materialized query tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO MATERIALIZED QUERY TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate materialized query tables of the tables in the invocation list, and to all immediate materialized query tables that are descendants of the descendent foreign key tables.

TO FOREIGN KEY TABLES
Specifies that the set integrity pending state is to be immediately cascaded to descendent foreign key tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state.

TO STAGING TABLES
Specifies that the set integrity pending state is to be immediately cascaded to descendent staging tables. Other descendent tables might later be put in set integrity pending state, if necessary, when the table is brought out of set integrity pending state. If both TO FOREIGN KEY TABLES and TO STAGING TABLES are specified, the set integrity pending state will be immediately cascaded to all descendent foreign key tables, all descendent immediate staging tables of the tables in the invocation list, and to all immediate staging tables that are descendants of the descendent foreign key tables.

CASCADE DEFERRED
Specifies that only the tables in the invocation list are to be put in set integrity pending state. The states of the descendent tables will remain unchanged. Descendent foreign key tables might later be implicitly put in set integrity pending state when their parent tables are checked for constraints violations. Descendent immediate materialized query tables and descendent immediate staging tables might be implicitly put in set integrity pending state when one of their underlying tables is checked for integrity violations.

If *cascade-clause* is not specified, the set integrity pending state is immediately cascaded to all descendent tables.

IMMEDIATE CHECKED
Specifies that the table is to be taken out of set integrity pending state by performing required integrity processing on the table. This is done in accordance with the information set in the STATUS and CONST_CHECKED columns of the SYSCAT.TABLES catalog view. That is:

- The value in the STATUS column must be ’C’ (the table is in set integrity pending state), or an error is returned (SQLSTATE 51027), unless the table is a descendent foreign key table, descendent materialized query table, or descendent staging table of a table that is specified in the list, is in set integrity pending state, and whose intermediate ancestors are also in the list.
- If the table being checked is in set integrity pending state, the value in CONST_CHECKED indicates which integrity options are to be checked.

When the table is taken out of set integrity pending state, its descendent tables are, if necessary, put in set integrity pending state. A warning to indicate that descendent tables have been put in set integrity pending state is returned (SQLSTATE 01586).

If the table is a system-maintained materialized query table, the data is checked against the query and refreshed as necessary. (IMMEDIATE CHECKED cannot be used for user-maintained materialized query tables.) If the table is a staging table, the data is checked against its query definition and propagated as necessary.

When the integrity of a child table is checked:

- None of its parents can be in set integrity pending state, or
- Each of its parents must be checked for constraints violations in the same SET INTEGRITY statement

When an immediate materialized query table is refreshed, or deltas are propagated to a staging table:

- None of its underlying tables can be in set integrity pending state, or

- Each of its underlying tables must be checked in the same SET INTEGRITY statement

Otherwise, an error is returned (SQLSTATE 428A8).

table-checked-options

online-options
Specifies the accessibility of the table while it is being processed.

ALLOW NO ACCESS
Specifies that no other users can access the table while it is being processed.

ALLOW READ ACCESS
Specifies that other users have read-only access to the table while it is being processed.

ALLOW WRITE ACCESS
Specifies that other users have read and write access to the table while it is being processed.

GENERATE IDENTITY
Specifies that if the table includes an identity column, the values are generated by the SET INTEGRITY statement. By default, when the GENERATE IDENTITY option is specified, only attached rows will have their identity column values generated by the SET INTEGRITY statement. The NOT INCREMENTAL option must be specified in conjunction with the GENERATE IDENTITY option to have the SET INTEGRITY statement generate identity column values for all rows in the table, including attached rows, loaded rows, and existing rows. If the GENERATE IDENTITY option is not specified, the current identity column values for all rows in the table are left unchanged.

query-optimization-options
Specifies the query optimization options for the maintenance of REFRESH DEFERRED materialized query tables.

ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY
Specifies that when the CURRENT REFRESH AGE special register is set to 'ANY', the maintenance of *table-name* will allow REFRESH DEFERRED materialized query tables to be used to optimize the query that maintains *table-name*. If *table-name* is not a REFRESH DEFERRED materialized query table, an error is returned (SQLSTATE 428FH). REFRESH IMMEDIATE materialized query tables are always considered during query optimization.

check-options

incremental-options

INCREMENTAL
Specifies the application of integrity processing on the appended portion (if any) of the table. If such a request cannot be satisfied (that is, the system detects that the whole table needs to be checked for data integrity), an error is returned (SQLSTATE 55019).

NOT INCREMENTAL
Specifies the application of integrity processing on the whole table. If the table is a materialized query table, the materialized query table definition is recomputed. If the table has at least one

constraint defined on it, this option causes full processing of descendent foreign key tables and descendent immediate materialized query tables. If the table is a staging table, it is set to an inconsistent state.

If the *incremental-options* clause is not specified, the system determines whether incremental processing is possible; if not, the whole table is checked.

FORCE GENERATED
If the table includes generated by expression columns, the values are computed on the basis of the expression and stored in the column. If this option is not specified, the current values are compared to the computed value of the expression, as though an equality check constraint were in effect. If the table is processed for integrity incrementally, generated columns are computed only for the appended portion.

PRUNE
This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH). If the INCREMENTAL check option is also specified, an error is returned (SQLSTATE 428FH).

FULL ACCESS
Specifies that the table is to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table (that has dependent immediate materialized query tables or dependent immediate staging tables) in the invocation list is incrementally processed, the underlying table is put in no data movement state, as required, after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables are taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE CHECKED option, the underlying table is put directly in full access state (bypassing the no data movement state). Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them might be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is put directly into full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option was specified.

exception-clause

FOR EXCEPTION
Specifies that any row that is in violation of a constraint being checked is to be moved to an exception table. Even if errors are

detected, the table is taken out of set integrity pending state. A warning to indicate that one or more rows have been moved to the exception tables is returned (SQLSTATE 01603).

If the FOR EXCEPTION option is not specified and any constraints are violated, only the first detected violation is returned (SQLSTATE 23514). If there is a violation in any table, all of the tables are left in set integrity pending state.

It is recommended to always use the FOR EXCEPTION option when checking for constraints violations to prevent a rollback of the SET INTEGRITY statement if a violation is found.

IN *table-name*
Specifies the table from which rows that violate constraints are to be moved. There must be one exception table specified for each table being checked. This clause cannot be specified for a materialized query table or a staging table (SQLSTATE 428A7).

USE *table-name*
Specifies the exception table into which error rows are to be moved.

FULL ACCESS
If the FULL ACCESS option is specified as the only operation of the statement, the table is placed into the full access state without being rechecked for integrity violations. However, dependent immediate materialized query tables that have not been refreshed might require a full recomputation in subsequent REFRESH TABLE statements, and dependent immediate staging tables that have not had the delta portions of the table propagated to them might be changed to incomplete state. This option can only be specified for a table that is in the no data movement state or the no access state, but not in the set integrity pending state (SQLSTATE 428FH).

PRUNE
This option can be specified for staging tables only. Specifies that the content of the staging table is to be pruned, and that the staging table is to be set to an inconsistent state. If any table in the *table-name* list is not a staging table, an error is returned (SQLSTATE 428FH).

table-unchecked-options

integrity-options
Used to define the types of required integrity processing that are to be bypassed when the table is taken out of the set integrity pending state.

ALL
The table will be immediately taken out of set integrity pending state without any of its required integrity processing being performed.

FOREIGN KEY
Required foreign key constraints checking will not be performed when the table is brought out of set integrity pending state.

CHECK
Required check constraints checking will not be performed when the table is brought out of set integrity pending state.

MATERIALIZED QUERY
Required refreshing of a materialized query table will not be performed when the table is brought out of set integrity pending state.

GENERATED COLUMN
Required generated column constraints checking will not be performed when the table is brought out of set integrity pending state.

STAGING
Required propagation of data to a staging table will not be performed when the table is brought out of set integrity pending state.

If no other types of integrity processing are required on the table after a specific type of integrity processing has been marked as bypassed, the table is immediately taken out of set integrity pending state.

FULL ACCESS
Specifies that the tables are to become fully accessible after the SET INTEGRITY statement executes.

When an underlying table in the invocation list is incrementally processed, and it has dependent immediate materialized query tables or dependent immediate staging tables, the underlying table is placed, as required, in the no data movement state after the SET INTEGRITY statement executes. When all incrementally refreshable dependent immediate materialized query tables and staging tables have been taken out of set integrity pending state, the underlying table is automatically brought out of the no data movement state into the full access state. If the FULL ACCESS option is specified with the IMMEDIATE UNCHECKED option, the underlying table is placed directly in full access state (it bypasses the no data movement state). Dependent immediate materialized query tables that have not been refreshed might undergo a full recomputation in the subsequent REFRESH TABLE statement, and dependent immediate staging tables that have not had the appended portions of the table propagated to them mIGHT be flagged as inconsistent.

When an underlying table in the invocation list requires full processing, or does not have dependent immediate materialized query tables, or dependent immediate staging tables, the underlying table is placed directly in full access state after the SET INTEGRITY statement executes, regardless of whether the FULL ACCESS option has been specified.

If the FULL ACCESS option has been specified with the IMMEDIATE UNCHECKED option, and the statement does not bring the table out of set integrity pending state, an error is returned (SQLSTATE 428FH).

IMMEDIATE UNCHECKED
Specifies one of the following:

- The table is to be brought out of set integrity pending state immediately without any required integrity processing.
- The table is to have one or more types of required integrity processing bypassed when the table is brought out of set integrity pending state by a subsequent SET INTEGRITY statement using the IMMEDIATE CHECKED option.

Consider the data integrity implications of this option before using it. See the “Notes” section below.

Notes:

- Effects on tables in one of the restricted set integrity-related states:
  - Use of INSERT, UPDATE, or DELETE is disallowed on a table that is in read access state or in no access state. Furthermore, any statement that requires

this type of modification to a table that is in such a state will be rejected. For example, deletion of a row in a parent table that cascades to a dependent table that is in the no access state is not allowed.

  - Use of SELECT is disallowed on a table that is in the no access state. Furthermore, any statement that requires read access to a table that is in the no access state will be rejected.
  - New constraints added to a table are normally enforced immediately. However, if the table is in set integrity pending state, the checking of any new constraints is deferred until the table is taken out of set integrity pending state. If the table is in set integrity pending state, addition of a new constraint places the table into set integrity pending no access state, because validity of data is at risk.
  - The CREATE INDEX statement cannot reference any table that is in read access state or in no access state. Similarly, an ALTER TABLE statement to add a primary key or a unique constraint cannot reference any table that is in read access state or in no access state.
  - The import utility is not allowed to operate on a table that is in read access state or in no access state.
  - The export utility is not allowed to operate on a table that is in no access state, but is allowed to operate on a table that is in read access state. If a table is in read access state, the export utility will only export the data that is in the non-appended portion.
  - Operations (like REORG, REDISTRIBUTE, update distribution key, update multidimensional clustering key, update range clustering key, update table partitioning key, and so on) that might involve data movement within a table are not allowed on a table that is in any of the following states: read access, no access, or no data movement.
  - The load, backup, restore, update statistics, runstats, reorgchk, list history, and rollforward utilities are allowed on a table that is in any of the following states: full access, read access, no access, or no data movement.
  - The ALTER TABLE, COMMENT, DROP TABLE, CREATE ALIAS, CREATE TRIGGER, CREATE VIEW, GRANT, REVOKE, and SET INTEGRITY statements can reference a table that is in any of the following states: full access, read access, no access, or no data movement. However, they might cause the table to be put into no access state.
  - Packages, views, and any other objects that depend on a table that is in no access state will return an error when the table is accessed at run time. Packages that depend on a table that is in read access state will return an error when an insert, update, or delete operation is attempted on the table at run time.

  The removal of violating rows by the SET INTEGRITY statement is not a delete event. Therefore, triggers are never activated by a SET INTEGRITY statement. Similarly, updating generated columns using the FORCE GENERATED option does not activate triggers.

- Incremental processing will be used whenever the situation allows it, because it is more efficient. The INCREMENTAL option is not needed in most cases. It is needed, however, to ensure that integrity checks are indeed processed incrementally. If the system detects that full processing is needed to ensure data integrity, an error is returned (SQLSTATE 55019).
- Warning about the use of the IMMEDIATE UNCHECKED clause:
  - This clause is intended to be used by utility programs, and its use by application programs is not recommended. If there is data in the table that

does not meet the integrity specifications that were defined for the table, and the IMMEDIATE UNCHECKED option is used, incorrect query results might be returned.

The fact that the table was taken out of the set integrity pending state without performing the required integrity processing will be recorded in the catalog (the respective byte in the CONST_CHECKED column in the SYSCAT.TABLES view will be set to 'U'). This indicates that the user has assumed responsibility for data integrity with respect to the specific constraints. This value remains unchanged until either:

- The table is put back into set integrity pending state (by referencing the table in a SET INTEGRITY statement with the OFF option), at which time 'U' values in the CONST_CHECKED column are changed to 'W' values, indicating that the user had previously assumed responsibility for data integrity, and the system needs to verify the data.
- All unchecked constraints for the table are dropped.

The 'W' state differs from the 'N' state in that it records the fact that integrity was previously checked by the user, but not yet by the system. If the user issues the SET INTEGRITY ... IMMEDIATE CHECKED statement with the NOT INCREMENTAL option, the system rechecks the whole table for data integrity (or performs a full refresh on a materialized query table), and then changes the 'W' state to the 'Y' state. If IMMEDIATE UNCHECKED is specified, or if NOT INCREMENTAL is not specified, the 'W' state is changed back to the 'U' state to record the fact that some data has still not been verified by the system. In the latter case (when the NOT INCREMENTAL is not specified), a warning is returned (SQLSTATE 01636).

If an underlying table's integrity has been checked using the IMMEDIATE UNCHECKED clause, the 'U' values in the CONST_CHECKED column of the underlying table will be propagated to the corresponding CONST_CHECKED column of:

- Dependent immediate materialized query tables
- Dependent deferred materialized query tables
- Dependent staging tables

For a dependent immediate materialized query table, this propagation is done whenever the underlying table is brought out of set integrity pending state, and whenever the materialized query table is refreshed. For a dependent deferred materialized query table, this propagation is done whenever the materialized query table is refreshed. For dependent staging tables, this propagation is done whenever the underlying table is brought out of set integrity pending state. These propagated 'U' values in the CONST_CHECKED columns of dependent materialized query tables and staging tables record the fact that these materialized query tables and staging tables depend on some underlying table whose required integrity processing has been bypassed using the IMMEDIATE UNCHECKED option.

For a materialized query table, the 'U' value in the CONST_CHECKED column that was propagated by the underlying table will remain until the materialized query table is fully refreshed and none of its underlying tables have a 'U' value in their corresponding CONST_CHECKED column. After such a refresh, the 'U' value in the CONST_CHECKED column for the materialized query table will be changed to 'Y'.

For a staging table, the 'U' value in the CONST_CHECKED column that was propagated by the underlying table will remain until the corresponding

deferred materialized query table of the staging table is refreshed. After such a refresh, the 'U' value in the CONST_CHECKED column for the staging table will be changed to 'Y'.
  - If a child table and its parent table are checked in the same SET INTEGRITY statement with the IMMEDIATE CHECKED option, and the parent table requires full checking of its constraints, the child table will have its foreign key constraints checked, independently of whether or not the child table has a 'U' value in the CONST_CHECKED column for foreign key constraints.
- After appending data using LOAD INSERT or ALTER TABLE ATTACH, the SET INTEGRITY statement with the IMMEDIATE CHECKED option checks the table for constraints violations. The system determines whether incremental processing on the table is possible. If so, only the appended portion is checked for integrity violations. If not, the system checks the whole table for integrity violations.
- Consider the statement:

  ```
  SET INTEGRITY FOR T IMMEDIATE CHECKED
  ```

  Situations in which the system will require a full refresh, or will check the whole table for integrity (the INCREMENTAL option cannot be specified) are:
  - When new constraints have been added to T itself while it is in the set integrity pending state
  - When a LOAD REPLACE operation against T, it parents, or its underlying tables has taken place
  - When the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on T, its parents, or its underlying tables
  - The cascading effect of full processing, when any parent of T (or underlying table, if T is a materialized query table or a staging table) has been checked for integrity non-incrementally
  - If the table space containing the table or its parent (or underlying table of a materialized query table or a staging table) has been rolled forward to a point in time, and the table and its parent (or underlying table if the table is a materialized query table or a staging table) reside in different table spaces
  - When T is a materialized query table, and a LOAD REPLACE or LOAD INSERT operation directly into T has taken place after the last refresh
- If the conditions for full processing described in the previous bullet are not satisfied, the system will attempt to check only the appended portion for integrity, or perform an incremental refresh (if it is a materialized query table) when the user does not specify the NOT INCREMENTAL option for the statement `SET INTEGRITY FOR T IMMEDIATE CHECKED`.
- If an error occurs during integrity processing, all the effects of the processing (including deleting from the original and inserting into the exception tables) will be rolled back.
- If a SET INTEGRITY statement issued with the FORCE GENERATED option fails because of a lack of log space, increase available active log space and reissue the SET INTEGRITY statement. Alternatively, use the SET INTEGRITY statement with the GENERATED COLUMN and IMMEDIATE UNCHECKED options to bypass generated column checking for the table. Then, issue a SET INTEGRITY statement with the IMMEDIATE CHECKED option and without the FORCE GENERATED option to check the table for other integrity violations (if applicable) and to bring it out of set integrity pending state. After the table is out of the set integrity pending state, the generated columns can be updated to their default (generated) values by assigning them to the keyword DEFAULT in

an UPDATE statement. This is accomplished by using either multiple searched update statements based on ranges (each followed by a commit), or a cursor-based approach using intermittent commits. A “with hold” cursor should be used if locks are to be retained after intermittent commits using the cursor-based approach.

- A table that was put into set integrity pending state using the CASCADE DEFERRED option of the SET INTEGRITY statement or the LOAD command, or through the ALTER TABLE statement with the ATTACH clause, and that is checked for integrity violations using the IMMEDIATE CHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the entire table is checked for integrity violations, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table is checked for integrity violations incrementally, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table requires no checking at all, its descendent immediate materialized query tables, descendent staging tables, and descendent foreign key tables will remain in their original states.
- A table that was put in set integrity pending state using the CASCADE DEFERRED option (of the SET INTEGRITY statement or the LOAD command), and that is brought out of set integrity pending state using the IMMEDIATE UNCHECKED option of the SET INTEGRITY statement, will have its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables put in set integrity pending state, as required:
  - If the table has been loaded using the REPLACE mode, its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables will be put in set integrity pending state.
  - If the table has been loaded using the INSERT mode, its descendent immediate materialized query tables and staging tables will be put in set integrity pending state, and its descendent foreign key tables will remain in their original states.
  - If the table has not been loaded, its descendent immediate materialized query tables, descendent staging tables, and its descendent foreign key tables will remain in their original states.
- SET INTEGRITY is usually a long running statement. In light of this, to reduce the risk of a rollback of the entire statement because of a lock timeout, you can issue the SET CURRENT LOCK TIMEOUT statement with the WAIT option before executing the SET INTEGRITY statement, and then reset the special register to its previous value after the transaction commits. Note, however, that the CURRENT LOCK TIMEOUT special register only impacts a specific set of lock types.
- If you use the ALLOW QUERY OPTIMIZATION USING REFRESH DEFERRED TABLES WITH REFRESH AGE ANY option, ensure that the maintenance order is correct for REFRESH DEFERRED materialized query tables. For example, consider two materialized query tables, MQT1 and MQT2, whose materialized queries share the same underlying tables. The materialized query for MQT2 can be calculated using MQT1, instead of the underlying tables. If separate statements are used to maintain these two materialized query tables, and MQT2

is maintained first, the system might choose to use the contents of MQT1, which has not yet been maintained, to maintain MQT2. In this case, MQT1 would contain current data, but MQT2 could still contain stale data, even though both were maintained at almost the same time. The correct maintenance order, if two SET INTEGRITY statements are used instead of one, is to maintain MQT1 first.

- When using the SET INTEGRITY statement to perform integrity processing on a base table that has been loaded or attached, it is recommended that you process its dependent REFRESH IMMEDIATE materialized query tables and its PROPAGATE IMMEDIATE staging tables in the same SET INTEGRITY statement to avoid putting these dependent tables in set integrity pending no access state at the end of SET INTEGRITY processing. Note that for base tables that have a large number of dependent REFRESH IMMEDIATE materialized query tables and PROPAGATE IMMEDIATE staging tables, memory constraints might make it impossible to process all of the dependents in the same statement as the base table.
- If the FORCE GENERATED or the GENERATE IDENTITY option is specified, and the column that is generated is part of a unique index, the SET INTEGRITY statement returns an error (SQLSTATE 23505) and rolls back if it detects duplicate keys in the unique index. This error is returned even if there is an exception table for the table being processed.

  This scenario can occur under the following circumstances:

  – The SET INTEGRITY statement runs after a LOAD command against the table, and the GENERATEDOVERRIDE or the IDENTITYOVERRIDE file type modifier is specified during the load operation. To prevent this scenario, it is recommended that you use the GENERATEDIGNORE or the GENERATEDMISSING file type modifer instead of GENERATEDOVERRIDE, and that you use the IDENTITYIGNORE or the IDENTITYMISSING modifier instead of IDENTITYOVERRIDE. Using the recommended modifiers will prevent the need for any generated by expression column or identity column processing during SET INTEGRITY statement execution.

  – The SET INTEGRITY statement is run after an ALTER TABLE statement that alters the expression of a generated by expression column.

  To bring a table out of the set integrity pending state after encountering such a scenario:

  – Do not use the FORCE GENERATED or the GENERATE IDENTITY option to regenerate the column values. Instead, use the IMMEDIATE CHECKED option in conjunction with the FOR EXCEPTION option to move any rows that violate the generated column expression to an exception table. Then, re-insert the rows into the table from the exception table, which will generate the correct expression and perform unique key checking. This prevents having to reprocess the entire table, because only those rows that violated the generated column expression will need to be processed again.

  – If the table being processed has attached partitions, detach those partitions before performing the actions that are described in the previous bullet. Then, re-attach the partitions and execute a SET INTEGRITY statement to process integrity on the attached partitions seperately.
- If a protected table is specified for the SET INTEGRITY statement along with an exception table, all of the following table criteria must be met; otherwise, an error is returned (SQLSTATE 428A5):

  – The tables must be protected by the same security policy.

  – If a column in the protected table has data type DB2SECURITYLABEL, the corresponding column in the exception table must also have data type DB2SECURITYLABEL.

– If a column in the protected table is protected by a security label, the corresponding column in the exception table must also be protected by the same security label.

- *Compatibilities*
  – For compatibility with previous versions of DB2:
    - SET CONSTRAINTS can be specified in place of SET INTEGRITY
    - SUMMARY can be specified in place of MATERIALIZED QUERY

Examples:

Example 1: The following is an example of a query that provides information about the set integrity pending state and the set integrity-related access restriction states of tables. SUBSTR is used to extract individual bytes of the CONST_CHECKED column of SYSCAT.TABLES. The first byte represents foreign key constraints; the second byte represents check constraints; the fifth byte represents materialized query table integrity; the sixth byte represents generated column constraints; the seventh byte represents staging table integrity; and the eighth byte represents data partitioning constraints. STATUS gives the set integrity pending state, and ACCESS_MODE gives the set integrity-related access restriction state.

```
SELECT TABNAME, STATUS, ACCESS_MODE,
  SUBSTR(CONST_CHECKED,1,1) AS FK_CHECKED,
  SUBSTR(CONST_CHECKED,2,1) AS CC_CHECKED,
  SUBSTR(CONST_CHECKED,5,1) AS MQT_CHECKED,
  SUBSTR(CONST_CHECKED,6,1) AS GC_CHECKED,
  SUBSTR(CONST_CHECKED,7,1) AS STG_CHECKED,
  SUBSTR(CONST_CHECKED,8,1) AS DP_CHECKED
FROM SYSCAT.TABLES
```

Example 2: Put the PARENT table in set integrity pending no access state, and immediately cascade the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
  NO ACCESS CASCADE IMMEDIATE
```

Example 3: Put the PARENT table in set integrity pending read access state without immediately cascading the set integrity pending state to its descendants.

```
SET INTEGRITY FOR PARENT OFF
  READ ACCESS CASCADE DEFERRED
```

Example 4: Check integrity for a table named FACT_TABLE. If there are no integrity violations detected, the table is brought out of set integrity pending state. If any integrity violations are detected, the entire statement is rolled back, and the table remains in set integrity pending state.

```
SET INTEGRITY FOR FACT_TABLE IMMEDIATE CHECKED
```

Example 5: Check integrity for the SALES and PRODUCTS tables, and move the rows that violate integrity into exception tables named SALES_EXCEPTIONS and PRODUCTS_EXCEPTIONS. Both the SALES and PRODUCTS tables are brought out of set integrity pending state, whether or not there are any integrity violations.

```
SET INTEGRITY FOR SALES, PRODUCTS IMMEDIATE CHECKED
  FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS,
  IN PRODUCTS USE PRODUCTS_EXCEPTIONS
```

Example 6: Enable FOREIGN KEY constraint checking in the MANAGER table, and CHECK constraint checking in the EMPLOYEE table, to be bypassed with the IMMEDIATE UNCHECKED option.

```
SET INTEGRITY FOR MANAGER FOREIGN KEY,
  EMPLOYEE CHECK IMMEDIATE UNCHECKED
```

Example 7: Add a check constraint and a foreign key to the EMP_ACT table, using two ALTER TABLE statements. The SET INTEGRITY statement with the OFF option is used to put the table in set integrity pending state, so that the constraints are not checked immediately upon execution of the two ALTER TABLE statements. The single SET INTEGRITY statement with the IMMEDIATE CHECKED option is used to check both of the added constraints during a single pass through the table.

```
SET INTEGRITY FOR EMP_ACT OFF;
ALTER TABLE EMP_ACT ADD CHECK
  (EMSTDATE <= EMENDATE);
ALTER TABLE EMP_ACT ADD FOREIGN KEY
  (EMPNO) REFERENCES EMPLOYEE;
SET INTEGRITY FOR EMP_ACT IMMEDIATE CHECKED
  FOR EXCEPTION IN EMP_ACT USE EMP_ACT_EXCEPTIONS
```

Example 8: Update generated columns with the correct values.

```
SET INTEGRITY FOR SALES IMMEDIATE CHECKED
  FORCE GENERATED
```

Example 9: Append (using LOAD INSERT) from different sources into an underlying table (SALES) of a REFRESH IMMEDIATE materialized query table (SALES_SUMMARY). Check SALES incrementally for data integrity, and refresh SALES_SUMMARY incrementally. In this scenario, integrity checking for SALES and refreshing of SALES_SUMMARY are incremental, because the system chooses incremental processing. The ALLOW READ ACCESS option is used on the SALES table to allow concurrent reads of existing data while integrity checking of the loaded portion of the table is taking place.

```
LOAD FROM 2000_DATA.DEL OF DEL
  INSERT INTO SALES ALLOW READ ACCESS;
LOAD FROM 2001_DATA.DEL OF DEL
  INSERT INTO SALES ALLOW READ ACCESS;
SET INTEGRITY FOR SALES ALLOW READ ACCESS IMMEDIATE CHECKED
  FOR EXCEPTION IN SALES USE SALES_EXCEPTIONS;
REFRESH TABLE SALES_SUMMARY;
```

Example 10: Attach a new partition to a data partitioned table named SALES. Incrementally check for constraints violations in the attached data of the SALES table and incrementally refresh the dependent SALES_SUMMARY table. The ALLOW WRITE ACCESS option is used on both tables to allow concurrent updates while integrity checking is taking place.

```
ALTER TABLE SALES
  ATTACH PARTITION STARTING (100) ENDING (200)
  FROM SOURCE;
SET INTEGRITY FOR SALES ALLOW WRITE ACCESS, SALES_SUMMARY ALLOW WRITE ACCESS
  IMMEDIATE CHECKED FOR EXCEPTION IN SALES
  USE SALES_EXCEPTIONS;
```

Example 11: Detach a partition from a data partitioned table named SALES. Incrementally refresh the dependent SALES_SUMMARY table.

```
ALTER TABLE SALES
  DETACH PARTITION 2000_PART INTO ARCHIVE_TABLE;
SET INTEGRITY FOR SALES_SUMMARY
  IMMEDIATE CHECKED;
```

Related reference:

- "Exception tables" in *SQL Reference, Volume 1*

Related samples:
- "TbGenCol.java -- How to use generated columns (JDBC)"

SET SCHEMA

The SET SCHEMA statement changes the value of the CURRENT SCHEMA special register. It is not under transaction control. If the package is bound with the DYNAMICRULES BIND option, this statement does not affect the qualifier used for unqualified database object references.

Invocation:

The statement can be embedded in an application program or issued interactively. It is an executable statement that can be dynamically prepared.

Authorization:

None required.

Syntax:

```
           ┌─CURRENT─┐            ┌─=─┐
►►──SET────┴─────────┴──SCHEMA────┴───┴──┬─schema-name─────┬──────────────────────►◄
                                         ├─USER────────────┤
                                         ├─SESSION_USER────┤
                                         ├─SYSTEM_USER─────┤
                                         ├─CURRENT_USER────┤
                                         ├─host-variable───┤
                                         └─string-constant─┘
```

Description:

schema-name
: This one-part name identifies a schema that exists at the application server. The length must not exceed 30 bytes (SQLSTATE 42815). No validation that the schema exists is made at the time that the schema is set. If a *schema-name* is misspelled, it will not be caught, and it could affect the way subsequent SQL operates.

USER
: The value in the USER special register.

SESSION_USER
: The value in the SESSION_USER special register.

SYSTEM_USER
: The value in the SYSTEM_USER special register.

CURRENT_USER
: The value in the CURRENT_USER special register.

host-variable
: A variable of type CHAR or VARCHAR. The length of the contents of the *host-variable* must not exceed 30 (SQLSTATE 42815). It cannot be set to null. If *host-variable* has an associated indicator variable, the value of that indicator variable must not indicate a null value (SQLSTATE 42815).

The characters of the *host-variable* must be left justified. When specifying the *schema-name* with a *host-variable*, all characters must be specified in the exact case intended as there is no conversion to uppercase characters.

string-constant
A character string constant with a maximum length of 30.

Rules:

- If the value specified does not conform to the rules for a *schema-name*, an error (SQLSTATE 3F000) is raised.
- The value of the CURRENT SCHEMA special register is used as the schema name in all dynamic SQL statements, with the exception of the CREATE SCHEMA statement, where an unqualified reference to a database object exists.
- The QUALIFIER bind option specifies the schema name for use as the qualifier for unqualified database object names in static SQL statements.

Notes:

- The initial value of the CURRENT SCHEMA special register is equivalent to USER.
- Setting the CURRENT SCHEMA special register does not effect the CURRENT PATH special register. Hence, the CURRENT SCHEMA will not be included in the SQL path and functions, procedures and user-defined type resolution may not find these objects. To include the current schema value in the SQL path, whenever the SET SCHEMA statement is issued, also issue the SET PATH statement including the schema name from the SET SCHEMA statement.
- CURRENT SQLID is accepted as a synonym for CURRENT SCHEMA and the effect of a SET CURRENT SQLID statement will be identical to that of a SET CURRENT SCHEMA statement. No other effects, such as statement authorization changes, will occur.

Examples:

Example 1: The following statement sets the CURRENT SCHEMA special register.

```
SET SCHEMA RICK
```

Example 2: The following example retrieves the current value of the CURRENT SCHEMA special register into the host variable called CURSCHEMA.

```
EXEC SQL VALUES (CURRENT SCHEMA) INTO :CURSCHEMA;
```

The value would be RICK, set by the previous example.

SQL queries

A *query* specifies a result table. A query is a component of certain SQL statements. The three forms of a query are:

- subselect
- fullselect
- select-statement.

Authorization

For each table, view, or nickname referenced in the query, the authorization ID of the statement must have at least one of the following:

- SYSADM or DBADM authority
- CONTROL privilege
- SELECT privilege.

Group privileges, with the exception of PUBLIC, are not checked for queries that are contained in static SQL statements.

For nicknames, authorization requirements of the data source for the object referenced by the nickname are applied when the query is processed. The authorization ID of the statement may be mapped to a different authorization ID at the data source.

Related reference:
- "SELECT INTO statement" in *SQL Reference, Volume 2*

Subselect

```
►►──select-clause──from-clause──┬──────────────┬──┬─────────────────┬──────────────►
                                └─where-clause─┘  └─group-by-clause─┘

►──┬───────────────┬──┬─────────────────┬──┬────────────────────┬─────────────────►◄
   └─having-clause─┘  └─order-by-clause─┘  └─fetch-first-clause─┘
```

The *subselect* is a component of the fullselect.

A subselect specifies a result table derived from the tables, views or nicknames identified in the FROM clause. The derivation can be described as a sequence of operations in which the result of each operation is input for the next. (This is only a way of describing the subselect. The method used to perform the derivation can be quite different from this description. If portions of the subselect do not actually need to be executed for the correct result to be obtained, they might or might not be executed.)

The clauses of the subselect are processed in the following sequence:
1. FROM clause
2. WHERE clause
3. GROUP BY clause
4. HAVING clause
5. SELECT clause
6. ORDER BY clause
7. FETCH FIRST clause

A subselect that contains an ORDER BY or FETCH FIRST clause cannot be specified:
- In the outermost fullselect of a view.
- In a materialized query table.
- Unless the subselect is enclosed in parenthesis.

For example, the following is not valid (SQLSTATE 428FJ):

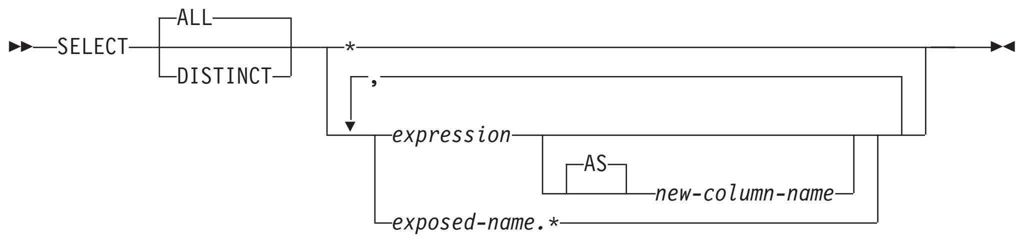
```
SELECT * FROM T1
   ORDER BY C1
UNION
SELECT * FROM T2
   ORDER BY C1
```

The following example *is* valid:

```
(SELECT * FROM T1
   ORDER BY C1)
UNION
(SELECT * FROM T2
   ORDER BY C1)
```

Note: An ORDER BY clause in a subselect does not affect the order of the rows returned by a query. An ORDER BY clause only affects the order of the rows returned if it is specified in the outermost fullselect.

select-clause



The SELECT clause specifies the columns of the final result table. The column values are produced by the application of the *select list* to R. The select list is the names or expressions specified in the SELECT clause, and R is the result of the previous operation of the subselect. For example, if the only clauses specified are SELECT, FROM, and WHERE, R is the result of that WHERE clause.

ALL
Retains all rows of the final result table, and does not eliminate redundant duplicates. This is the default.

DISTINCT
Eliminates all but one of each set of duplicate rows of the final result table. If DISTINCT is used, no string column of the result table can be a LONG VARCHAR, LONG VARGRAPHIC, DATALINK, LOB type, distinct type on any of these types, or structured type. DISTINCT may be used more than once in a subselect. This includes SELECT DISTINCT, the use of DISTINCT in a column function of the select list or HAVING clause, and subqueries of the subselect.

Two rows are duplicates of one another only if each value in the first is equal to the corresponding value of the second. For determining duplicates, two null values are considered equal.

Select list notation:

* Represents a list of names that identify the columns of table R. The first name in the list identifies the first column of R, the second name identifies the second column of R, and so on.

The list of names is established when the program containing the SELECT clause is bound. Hence * (the asterisk) does not identify any columns that have been added to a table after the statement containing the table reference has been bound.

expression
: Specifies the values of a result column. Can be any expression that is a valid SQL language element, but commonly includes column names. Each column name used in the select list must unambiguously identify a column of R.

new-column-name or **AS** *new-column-name*
: Names or renames the result column. The name must not be qualified and does not have to be unique. Subsequent usage of column-name is limited as follows:

- A new-column-name specified in the AS clause can be used in the order-by-clause, provided the name is unique.
- A new-column-name specified in the AS clause of the select list cannot be used in any other clause within the subselect (where-clause, group-by-clause or having-clause).
- A new-column-name specified in the AS clause cannot be used in the update-clause.
- A new-column-name specified in the AS clause is known outside the fullselect of nested table expressions, common table expressions and CREATE VIEW.

*name.**
: Represents the list of names that identify the columns of the result table identified by *exposed-name*. The *exposed-name* may be a table name, view name, nickname, or correlation name, and must designate a table, view or nickname named in the FROM clause. The first name in the list identifies the first column of the table, view or nickname, the second name in the list identifies the second column of the table, view or nickname, and so on.

The list of names is established when the statement containing the SELECT clause is bound. Therefore, * does not identify any columns that have been added to a table after the statement has been bound.

The number of columns in the result of SELECT is the same as the number of expressions in the operational form of the select list (that is, the list established when the statement is prepared), and cannot exceed 500 for a 4K page size or 1012 for an 8K, 16K, or 32K page size.

Limitations on string columns

For limitations on the select list, see “Restrictions Using Varying-Length Character Strings”.

Applying the select list

Some of the results of applying the select list to R depend on whether or not GROUP BY or HAVING is used. The results are described in two separate lists:

If GROUP BY or HAVING is used:

- An expression *X* (not a column function) used in the select list must have a GROUP BY clause with:
  - a *grouping-expression* in which each column-name unambiguously identifies a column of R (see “group-by-clause” on page 1226) or
  - each column of R referenced in *X* as a separate *grouping-expression*.

- The select list is applied to each group of R, and the result contains as many rows as there are groups in R. When the select list is applied to a group of R, that group is the source of the arguments of the column functions in the select list.

If neither GROUP BY nor HAVING is used:

- Either the select list must not include any column functions, or each *column-name* in the select list must be specified within a column function or must be a correlated column reference.
- If the select does not include column functions, then the select list is applied to each row of R and the result contains as many rows as there are rows in R.
- If the select list is a list of column functions, then R is the source of the arguments of the functions and the result of applying the select list is one row.

In either case the *n*th column of the result contains the values specified by applying the *n*th expression in the operational form of the select list.

Null attributes of result columns: Result columns do not allow null values if they are derived from:

- A column that does not allow null values
- A constant
- The COUNT or COUNT_BIG function
- A host variable that does not have an indicator variable
- A scalar function or expression that does not include an operand that allows nulls

Result columns allow null values if they are derived from:

- Any column function except COUNT or COUNT_BIG
- A column that allows null values
- A scalar function or expression that includes an operand that allows nulls
- A NULLIF function with arguments containing equal values
- A host variable that has an indicator variable
- A result of a set operation if at least one of the corresponding items in the select list is nullable
- An arithmetic expression or view column that is derived from an arithmetic expression and the database is configured with DFT_SQLMATHWARN set to Yes
- A scalar subselect
- A dereference operation

Names of result columns:

- If the AS clause is specified, the name of the result column is the name specified on the AS clause.
- If the AS clause is not specified and the result column is derived from a column, then the result column name is the unqualified name of that column.
- If the AS clause is not specified and the result column is derived using a dereference operation, then the result column name is the unqualified name of the target column of the dereference operation.
- All other result column names are unnamed. The system assigns temporary numbers (as character strings) to these columns.

Data types of result columns: Each column of the result of SELECT acquires a data type from the expression from which it is derived.

When the expression is ...	The data type of the result column is ...
the name of any numeric column	the same as the data type of the column, with the same precision and scale for DECIMAL columns.
an integer constant	INTEGER.
a decimal constant	DECIMAL, with the precision and scale of the constant.
a floating-point constant	DOUBLE.
the name of any numeric variable	the same as the data type of the variable, with the same precision and scale for DECIMAL variables.
a hexadecimal constant representing n bytes	VARCHAR(n); the code page is the database code page.
the name of any string column	the same as the data type of the column, with the same length attribute.
the name of any string variable	the same as the data type of the variable, with the same length attribute; if the data type of the variable is not identical to an SQL data type (for example, a NUL-terminated string in C), the result column is a varying-length string.
a character string constant of length *n*	VARCHAR(*n*).
a graphic string constant of length *n*	VARGRAPHIC(*n*).
the name of a datetime column	the same as the data type of the column.
the name of a user-defined type column	the same as the data type of the column.
the name of a reference type column	the same as the data type of the column.

from-clause

```
             ┌─,───────────────┐
             ▼                 │
►►──FROM──────table-reference──┴──────────────────────────────────────────►◄
```

The FROM clause specifies an intermediate result table.

If one table-reference is specified, the intermediate result table is simply the result of that table-reference. If more than one table-reference is specified, the intermediate result table consists of all possible combinations of the rows of the specified table-references (the Cartesian product). Each row of the result is a row from the first table-reference concatenated with a row from the second table-reference, concatenated in turn with a row from the third, and so on. The number of rows in the result is the product of the number of rows in all the individual table-references. For a description of *table-reference*, see “table-reference” on page 1216.

table-reference

```
>>-+-table-name--+--------------------+--+---------------------+-+----------><
   |             '-| correlation-clause |-' '-| tablesample-clause |-' |
   +-+-+-nickname--+------------------+--+--------------------------+-+
   | | '-view-name-'                  |  '-| correlation-clause |-' |
   | '-+-ONLY--+--(--+-table-name-+--)-'                            |
   |   '-OUTER-'     '-view-name--'                                 |
   +-TABLE--(--function-name--(--+----------------+--)--)--| correlation-clause |-+
   |                             | .-,----------. |                              |
   |                             | V            | |                              |
   |                             '---expression-+-'                              |
   |                         (1)                                                 |
   +-xmltable-expression------+--| correlation-clause |-------------------------+
   +-| nested-table-expression |-'                                               |
   +-| data-change-table-reference |--+--------------------------+-----------------+
   |                                  '-| correlation-clause |-'                  |
   '-joined-table------------------------------------------------------------'
```

correlation-clause:

```
   .-AS-.
|--+----+--correlation-name--+-------------------------+--------------|
                             |    .-,-----------.      |
                             |    V             |      |
                             '-(----column-name-+--)---'
```

tablesample-clause:

```
|--TABLESAMPLE--+-BERNOULLI-+--(--numeric-expression1--)------------------>
                '-SYSTEM----'

>--+----------------------------------------+-----------------------------|
   '-REPEATABLE--(--numeric-expression2--)--'
```

nested-table-expression:

```
|--+-------------------------------------------------+--(fullselect)--| correlation-clause |--|
   |         (2)                                     |
   '-LATERAL--+-----------------------------------+--'
              '-| continue-handler |--WITHIN-------'
```

continue-handler:

```
                          .-,-----------------------------.
                          V                               |
|--RETURN DATA UNTIL--------| specific-condition-value |--+----------------|
```

specific-condition-value:

```
                              .-VALUE-.
|--FEDERATED--SQLSTATE--+-------+--string-constant--+-------------------------------+--|
                                                    |          .-,----------------. |
                                                    |          V                  | |
                                                    '-SQLCODE----integer-constant-+-'
```

data-change-table-reference:

```
├──┬─┬─FINAL─┬──TABLE──(──insert-statement──)─────────┬──────────────────────────┤
   │ └─NEW───┘                                        │
   ├─┬─FINAL─┬──TABLE──(──searched-update-statement──)─┤
   │ ├─NEW───┤                                        │
   │ └─OLD───┘                                        │
   └─OLD TABLE──(──searched-delete-statement──)───────┘
```

Notes:

1 An XMLTABLE expression can be part of a table-reference. In this case, subexpressions within the XMLTABLE expression are in-scope of prior range variables in the FROM clause. For more information, see the description of "XMLTABLE".

2 TABLE can be specified in place of LATERAL.

Each *table-name*, *view-name* or *nickname* specified as a table-reference must identify an existing table, view or nickname at the application server or the *table-name* of a common table expression defined preceding the fullselect containing the table-reference. If the *table-name* references a typed table, the name denotes the UNION ALL of the table with all its subtables, with only the columns of the *table-name*. Similarly, if the *view-name* references a typed view, the name denotes the UNION ALL of the view with all its subviews, with only the columns of the *view-name*.

The use of ONLY(*table-name*) or ONLY(*view-name*) means that the rows of the proper subtables or subviews are not included. If the *table-name* used with ONLY does not have subtables, then ONLY(*table-name*) is equivalent to specifying *table-name*. If the *view-name* used with ONLY does not have subviews, then ONLY(*view-name*) is equivalent to specifying *view-name*.

The use of OUTER(*table-name*) or OUTER(*view-name*) represents a virtual table. If the *table-name* or *view-name* used with OUTER does not have subtables or subviews, then specifying OUTER is equivalent to not specifying OUTER. OUTER(*table-name*) is derived from *table-name* as follows:

- The columns include the columns of *table-name* followed by the additional columns introduced by each of its subtables (if any). The additional columns are added on the right, traversing the subtable hierarchy in depth-first order. Subtables that have a common parent are traversed in creation order of their types.
- The rows include all the rows of *table-name* and all the rows of its subtables. Null values are returned for columns that are not in the subtable for the row.

The previous points also apply to OUTER(*view-name*), substituting *view-name* for *table-name* and subview for subtable.

The use of ONLY or OUTER requires the SELECT privilege on every subtable of *table-name* or subview of *view-name*.

Each *function-name* together with the types of its arguments, specified as a table reference must resolve to an existing table function at the application server.

A fullselect in parentheses followed by a correlation name is called a *nested table expression*.

A *joined-table* specifies an intermediate result set that is the result of one or more join operations. For more information, see "joined-table" on page 1224.

The exposed names of all table references should be unique. An exposed name is:
- A *correlation-name*,
- A *table-name* that is not followed by a *correlation-name*,
- A *view-name* that is not followed by a *correlation-name*,
- A *nickname* that is not followed by a *correlation-name*,
- An *alias-name* that is not followed by a *correlation-name*.

Each *correlation-name* is defined as a designator of the immediately preceding *table-name*, *view-name*, *nickname*, *function-name* reference or nested table expression. Any qualified reference to a column for a table, view, table function or nested table expression must use the exposed name. If the same table name, view or nickname name is specified twice, at least one specification should be followed by a *correlation-name*. The *correlation-name* is used to qualify references to the columns of the table, view or nickname. When a *correlation-name* is specified, *column-name*s can also be specified to give names to the columns of the *table-name*, *view-name*, *nickname*, *function-name* reference or nested table expression.

In general, table functions and nested table expressions can be specified on any from-clause. Columns from the table functions and nested table expressions can be referenced in the select list and in the rest of the subselect using the correlation name which must be specified. The scope of this correlation name is the same as correlation names for other table, view or nickname in the FROM clause. A nested table expression can be used:
- In place of a view to avoid creating the view (when general use of the view is not required)
- When the desired result table is based on host variables

An expression in the select list of a nested table expression that is referenced within, or is the target of, a data change statement within a fullselect is only valid when it does not include:
- A function that reads or modifies SQL data
- A function that is non-deterministic
- A function that has external action
- An OLAP function

If a view is referenced directly in, or as the target of a nested table expression in a data change statement within a FROM clause, the view must either be symmetric (have WITH CHECK OPTION specified) or satisfy the restriction for a WITH CHECK OPTION view.

If the target of a data change statement within a FROM clause is a nested table expression, the modified rows are not requalified, WHERE clause predicates are not re-evaluated, and ORDER BY or FETCH FIRST operations are not redone.

The optional tablesample-clause can be used to obtain a random subset (a sample) of the rows from the specified *table-name*, rather than the entire contents of that *table-name*, for this query. This sampling is in addition to any predicates that are specified in the *where-clause*. Unless the optional REPEATABLE clause is specified, each execution of the query will usually yield a different sample, except in degenerate cases where the table is so small relative to the sample size that any sample must return the same rows. The size of the sample is controlled by the *numeric-expression1* in parentheses, representing an approximate percentage (P) of the table to be returned. The method by which the sample is obtained is specified after the TABLESAMPLE keyword, and can be either BERNOULLI or SYSTEM. For both methods, the exact number of rows in the sample may be different for each execution of the query, but on average should be approximately P percent of the table, before any predicates further reduce the number of rows.

The *table-name* must be a stored table. It can be a materialized query table (MQT) name, but not a subselect or table expression for which an MQT has been defined, because there is no guarantee that the database manager will route to the MQT for that subselect.

Semantically, sampling of a table occurs before any other query processing, such as applying predicates or performing joins. Repeated accesses of a sampled table within a single execution of a query (such as in a nested-loop join or a correlated subquery) will return the same sample. More than one table may be sampled in a query.

BERNOULLI sampling considers each row individually. It includes each row in the sample with probability P/100 (where P is the value of *numeric-expression1*), and excludes each row with probability 1 - P/100, independently of the other rows. So if the *numeric-expression1* evaluated to the value 10, representing a ten percent sample, each row would be included with probability 0.1, and excluded with probability 0.9.

SYSTEM sampling permits the database manager to determine the most efficient manner in which to perform the sampling. In most cases, SYSTEM sampling applied to a *table-name* means that each page of *table-name* is included in the sample with probability P/100, and excluded with probability 1 - P/100. All rows on each page that is included qualify for the sample. SYSTEM sampling of a *table-name* generally executes much faster than BERNOULLI sampling, because fewer data pages need to be retrieved; however, it can often yield less accurate estimates for aggregate functions (SUM(SALES), for example), especially if the rows of *table-name* are clustered on any columns referenced in that query. The optimizer may in certain circumstances decide that it is more efficient to perform SYSTEM sampling as if it were BERNOULLI sampling, for example when a predicate on *table-name* can be applied by an index and is much more selective than the sampling rate P.

The *numeric-expression1* specifies the size of the sample to be obtained from *table-name*, expressed as a percentage. It must be a constant numeric expression that cannot contain columns, parameter markers, or host variables. The expression must evaluate to a positive number that is less than or equal to 100, but can be between 1 and 0. For example, a value of 0.01 represents one one-hundredth of a percent, meaning that 1 row in 10 000 would be sampled, on average. A *numeric-expression1* that evaluates to 100 is handled as if the tablesample-clause were not specified. If *numeric-expression1* evaluates to the null value, or to a value that is greater than 100 or less than 0, an error is returned (SQLSTATE 2202H).

It is sometimes desirable for sampling to be repeatable from one execution of the query to the next; for example, during regression testing or query "debugging". This can be accomplished by specifying the REPEATABLE clause. The REPEATABLE clause requires the specification of a *numeric-expression2* in parentheses, which serves the same role as the seed in a random number generator. Adding the REPEATABLE clause to the tablesample-clause of any *table-name* ensures that repeated executions of that query (using the same value for *numeric-expression2*) return the same sample, assuming, of course, that the data itself has not been updated, reorganized, or repartitioned. To guarantee that the same sample of *table-name* is used across multiple queries, use of a global temporary table is recommended. Alternatively, the multiple queries could be combined into one query, with multiple references to a sample that is defined using the WITH clause.

Following are some examples:

Example 1: Request a 10% Bernoulli sample of the Sales table for auditing purposes.

```
  SELECT * FROM Sales
    TABLESAMPLE BERNOULLI(10)
```

Example 2: Compute the total sales revenue in the Northeast region for each product category, using a random 1% SYSTEM sample of the Sales table. The semantics of SUM are for the sample itself, so to extrapolate the sales to the entire Sales table, the query must divide that SUM by the sampling rate (0.01).

```
SELECT SUM(Sales.Revenue) / (0.01)
  FROM Sales TABLESAMPLE SYSTEM(1)
  WHERE Sales.RegionName = 'Northeast'
  GROUP BY Sales.ProductCategory
```

Example 3: Using the REPEATABLE clause, modify the previous query to ensure that the same (yet random) result is obtained each time the query is executed. (The value of the constant enclosed by parentheses is arbitrary.)

```
SELECT SUM(Sales.Revenue) / (0.01)
  FROM Sales TABLESAMPLE SYSTEM(1) REPEATABLE(3578231)
  WHERE Sales.RegionName = 'Northeast'
  GROUP BY Sales.ProductCategory
```

Table function references

In general, a table function, together with its argument values, can be referenced in the FROM clause of a SELECT in exactly the same way as a table or view. There are, however, some special considerations which apply.

- Table Function Column Names

  Unless alternate column names are provided following the *correlation-name*, the column names for the table function are those specified in the RETURNS clause of the CREATE FUNCTION statement. This is analogous to the names of the columns of a table, which are defined in the CREATE TABLE statement.
- Table Function Resolution

  The arguments specified in a table function reference, together with the function name, are used by an algorithm called *function resolution* to determine the exact function to be used. This is no different from what happens with other functions (such as scalar functions) that are used in a statement.
- Table Function Arguments

  As with scalar function arguments, table function arguments can in general be any valid SQL expression. The following examples are valid syntax:

```
  Example 1:  SELECT c1
              FROM TABLE( tf1('Zachary') ) AS z
              WHERE c2 = 'FLORIDA';

  Example 2:  SELECT c1
              FROM TABLE( tf2 (:hostvar1, CURRENT DATE) ) AS z;

  Example 3:  SELECT c1
```

```
                FROM t
                WHERE c2 IN
                         (SELECT c3 FROM
                          TABLE( tf5(t.c4) ) AS z  -- correlated reference
                         )                         -- to previous FROM clause
```

- Table Functions That Modify SQL Data

  Table functions that are specified with the MODIFIES SQL DATA option can only be used as the last table reference in a *select-statement*, *common-table-expression*, or RETURN statement that is a subselect, a SELECT INTO, or a *row-fullselect* in a SET statement. Only one table function is allowed in one FROM clause, and the table function arguments must be correlated to all other table references in the subselect (SQLSTATE 429BL). The following examples have valid syntax for a table function with the MODIFIES SQL DATA property:

```
Example 1:  SELECT c1
            FROM TABLE( tfmod('Jones') ) AS z

Example 2:  SELECT c1
            FROM t1, t2, TABLE( tfmod(t1.c1, t2.c1) ) AS z

Example 3:  SET var =
            (SELECT c1
            FROM TABLE( tfmod('Jones') ) AS z

Example 4:  RETURN SELECT c1
            FROM TABLE( tfmod('Jones') ) AS z

Example 5:  WITH v1(c1) AS
            (SELECT c1
            FROM TABLE( tfmod(:hostvar1) ) AS z)
            SELECT c1
            FROM v1, t1 WHERE v1.c1 = t1.c1
```

Error tolerant nested-table-expression

Certain errors that occur within a *nested-table-expression* can be tolerated, and instead of returning an error, the query can continue and return a result.

Specifying the RETURN DATA UNTIL clause will cause any rows that are returned from the fullselect before the indicated condition is encountered to make up the result set from the fullselect. This means that a partial result set (which could also be an empty result set) from the fullselect is acceptable as the result for the *nested-table-expression*.

The FEDERATED keyword restricts the condition to handle only errors that occur at a remote data source.

The condition can be specified as an SQLSTATE value, with a *string-constant* length of 5. You can optionally specify an SQLCODE value for each specified SQLSTATE value. For portable applications, specify SQLSTATE values as much as possible, because SQLCODE values are generally not portable across platforms and are not part of the SQL standard.

Only certain conditions can be tolerated. Errors that do not allow the rest of the query to be executed cannot be tolerated, and an error is returned for the whole query. The *specific-condition-value* might specify conditions that cannot actually be tolerated by the database manager, even if a specific SQLSTATE or SQLCODE value is specified, and for these cases, an error is returned.

The following SQLSTATE values and SQLCODE values have the potential, when specified, to be tolerated by the database manager:

- SQLSTATE 08001; SQLCODEs -1336, -30080, -30081, -30082
- SQLSTATE 08004
- SQLSTATE 42501
- SQLSTATE 42704; SQLCODE -204
- SQLSTATE 42720
- SQLSTATE 28000

A query or view containing an error tolerant *nested-table-expression* is read-only.

The fullselect of an error tolerant *nested-table-expression* is not optimized using materialized query tables.

Correlated references in table-references

Correlated references can be used in nested table expressions or as arguments to table functions. The basic rule that applies for both these cases is that the correlated reference must be from a *table-reference* at a higher level in the hierarchy of subqueries. This hierarchy includes the table-references that have already been resolved in the left-to-right processing of the FROM clause. For nested table expressions, the TABLE keyword must appear before the fullselect. So the following examples are valid syntax:

```
Example 1:  SELECT t.c1, z.c5
            FROM t, TABLE( tf3(t.c2) ) AS z      -- t precedes tf3
            WHERE t.c3 = z.c4;                   -- in FROM, so t.c2
                                                 -- is known

Example 2:  SELECT t.c1, z.c5
            FROM t, TABLE( tf4(2 * t.c2) ) AS z  -- t precedes tf4
            WHERE t.c3 = z.c4;                   -- in FROM, so t.c2
                                                 -- is known

Example 3:  SELECT d.deptno, d.deptname,
                   empinfo.avgsal, empinfo.empcount
            FROM department d,
                 LATERAL (SELECT AVG(e.salary) AS avgsal,
                                 COUNT(*) AS empcount
                        FROM employee e           -- department precedes
                        WHERE e.workdept=d.deptno -- and TABLE is
                       ) AS empinfo;              -- specified, so
                                                  -- d.deptno is known
```

But the following examples are not valid:

```
Example 4:  SELECT t.c1, z.c5
            FROM TABLE( tf6(t.c2) ) AS z, t  -- cannot resolve t in t.c2!
            WHERE t.c3 = z.c4;               -- compare to Example 1 above.

Example 5:  SELECT a.c1, b.c5
            FROM TABLE( tf7a(b.c2) ) AS a, TABLE( tf7b(a.c6) ) AS b
            WHERE a.c3 = b.c4;               -- cannot resolve b in b.c2!

Example 6:  SELECT d.deptno, d.deptname,
                   empinfo.avgsal, empinfo.empcount
            FROM department d,
                 (SELECT AVG(e.salary) AS avgsal,
                                 COUNT(*) AS empcount
                        FROM employee e           -- department precedes
                        WHERE e.workdept=d.deptno -- but TABLE is not
                       ) AS empinfo;              -- specified, so
                                                  -- d.deptno is unknown
```

Data change table references

A *data-change-table-reference* clause specifies an intermediate result table. This table is based on the rows that are directly changed by the searched UPDATE, searched DELETE, or INSERT statement that is included in the clause. A *data-change-table-reference* can be specified as the only *table-reference* in the FROM clause of the outer fullselect that is used in a *select-statement*, a SELECT INTO statement, or a common table expression. A *data-change-table-reference* can be specified as the only table reference in the only fullselect in a SET Variable statement (SQLSTATE 428FL). The target table or view of the data change statement is considered to be a table or view that is referenced in the query; therefore, the authorization ID of the query must have SELECT privilege on that target table or view. A *data-change-table-reference* clause cannot be specified in a view definition, materialized query table definition, or FOR statement (SQLSTATE 428FL).

The target of the UPDATE, DELETE, or INSERT statement cannot be a temporary view defined in a common table expression (SQLSTATE 42807).

FINAL TABLE
Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they appear at the completion of the data change statement. If there are AFTER triggers or referential constraints that result in further operations on the table that is the target of the SQL data change statement, an error is returned (SQLSTATE 57058, SQLSTATE 560C6). If the target of the SQL data change statement is a view that is defined with an INSTEAD OF trigger for the type of data change, an error is returned (SQLSTATE 428G3).

NEW TABLE
Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement prior to the application of referential constraints and AFTER triggers. Data in the target table at the completion of the statement might not match the data in the intermediate result table because of additional processing for referential constraints and AFTER triggers.

OLD TABLE
Specifies that the rows of the intermediate result table represent the set of rows that are changed by the SQL data change statement as they existed prior to the application of the data change statement.

(searched-update-statement)
Specifies a searched UPDATE statement. A WHERE clause or a SET clause in the UPDATE statement cannot contain correlated references to columns outside of the UPDATE statement.

(searched-delete-statement)
Specifies a searched DELETE statement. A WHERE clause in the DELETE statement cannot contain correlated references to columns outside of the DELETE statement.

(insert-statement)
Specifies an INSERT statement. A fullselect in the INSERT statement cannot contain correlated references to columns outside of the fullselect of the INSERT statement.

The content of the intermediate result table for a *data-change-table-reference* is determined when the cursor opens. The intermediate result table contains all manipulated rows, including all the columns in the specified target table or view.

All the columns of the target table or view for an SQL data change statement are accessible using the column names from the target table or view. If an INCLUDE clause was specified within a data change statement, the intermediate result table will contain these additional columns.

joined-table

```
                                ┌─INNER─────┐
►►──┬─table-reference───┬──┼───────────┼──JOIN──table-reference──ON──join-condition──┬──────►◄
    │                   │  └─┤ outer ├─┘                                              │
    └─(──joined-table──)─┴────────────────────────────────────────────────────────────┘
```

outer:

```
                ┌─OUTER─┐
├───┬─LEFT──┬──┴───────┴──────────────────────────────────────────────┤
    ├─RIGHT─┤
    └─FULL──┘
```

A *joined table* specifies an intermediate result table that is the result of either an inner join or an outer join. The table is derived by applying one of the join operators: INNER, LEFT OUTER, RIGHT OUTER, or FULL OUTER to its operands.

Inner joins can be thought of as the cross product of the tables (combine each row of the left table with every row of the right table), keeping only the rows where the join condition is true. The result table may be missing rows from either or both of the joined tables. Outer joins include the inner join and preserve these missing rows. There are three types of outer joins:

- ***left outer join*** includes rows from the left table that were missing from the inner join.
- ***right outer join*** includes rows from the right table that were missing from the inner join.
- ***full outer join*** includes rows from both the left and right tables that were missing from the inner join.

If a join-operator is not specified, INNER is implicit. The order in which multiple joins are performed can affect the result. Joins can be nested within other joins. The order of processing for joins is generally from left to right, but based on the position of the required join-condition. Parentheses are recommended to make the order of nested joins more readable. For example:

```
TB1 LEFT JOIN TB2 ON TB1.C1=TB2.C1
  RIGHT JOIN TB3 LEFT JOIN TB4 ON TB3.C1=TB4.C1
    ON TB1.C1=TB3.C1
```

is the same as:

```
(TB1 LEFT JOIN TB2 ON TB1.C1=TB2.C1)
  RIGHT JOIN (TB3 LEFT JOIN TB4 ON TB3.C1=TB4.C1)
    ON TB1.C1=TB3.C1
```

A joined table can be used in any context in which any form of the SELECT statement is used. A view or a cursor is read-only if its SELECT statement includes a joined table.

A *join-condition* is a *search-condition*, except that:

- It cannot contain any subqueries, scalar or otherwise
- It cannot include any dereference operations or the DEREF function, where the reference value is other than the object identifier column
- It cannot include an SQL function
- Any column referenced in an expression of the *join-condition* must be a column of one of the operand tables of the associated join (in the scope of the same joined-table clause)
- Any function referenced in an expression of the *join-condition* of a full outer join must be deterministic and have no external action
- It cannot include an XMLQUERY or XMLEXISTS expression

An error occurs if the join condition does not comply with these rules (SQLSTATE 42972).

Column references are resolved using the rules for resolution of column name qualifiers. The same rules that apply to predicates apply to join conditions.

Join operations

A *join-condition* specifies pairings of T1 and T2, where T1 and T2 are the left and right operand tables of the JOIN operator of the *join-condition*. For all possible combinations of rows of T1 and T2, a row of T1 is paired with a row of T2 if the *join-condition* is true. When a row of T1 is joined with a row of T2, a row in the result consists of the values of that row of T1 concatenated with the values of that row of T2. The execution might involve the generation of a null row. The null row of a table consists of a null value for each column of the table, regardless of whether the columns allow null values.

The following summarizes the result of the join operations:

- The result of T1 INNER JOIN T2 consists of their paired rows where the join-condition is true.
- The result of T1 LEFT OUTER JOIN T2 consists of their paired rows where the join-condition is true and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T2 allow null values.
- The result of T1 RIGHT OUTER JOIN T2 consists of their paired rows where the join-condition is true and, for each unpaired row of T2, the concatenation of that row with the null row of T1. All columns derived from T1 allow null values.
- The result of T1 FULL OUTER JOIN T2 consists of their paired rows and, for each unpaired row of T2, the concatenation of that row with the null row of T1 and, for each unpaired row of T1, the concatenation of that row with the null row of T2. All columns derived from T1 and T2 allow null values.

where-clause

```
►►──WHERE──search-condition──────────────────────────────────────────►◄
```

The WHERE clause specifies an intermediate result table that consists of those rows of R for which the *search-condition* is true. R is the result of the FROM clause of the subselect.

The *search-condition* must conform to the following rules:

- Each *column-name* must unambiguously identify a column of R or be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a *table-reference* in an outer subselect.
- A column function must not be specified unless the WHERE clause is specified in a subquery of a HAVING clause and the argument of the function is a correlated reference to a group.

Any subquery in the *search-condition* is effectively executed for each row of R, and the results are used in the application of the *search-condition* to the given row of R. A subquery is actually executed for each row of R only if it includes a correlated reference. In fact, a subquery with no correlated references may be executed just once, whereas a subquery with a correlated reference may have to be executed once for each row.

group-by-clause

```
                  ┌─,──────────────────────┐
                  ▼                        │
►►──GROUP BY──────┬─grouping-expression─┬──┴─────────────────────────────►◄
                  ├─grouping-sets───────┤
                  └─super-groups────────┘
```

The GROUP BY clause specifies an intermediate result table that consists of a grouping of the rows of R. R is the result of the previous clause of the subselect.

In its simplest form, a GROUP BY clause contains a *grouping expression*. A grouping expression is an *expression* used in defining the grouping of R. Each *column name* included in grouping-expression must unambiguously identify a column of R (SQLSTATE 42702 or 42703). A grouping expression cannot include a scalar fullselect or an XMLQUERY or XMLEXISTS expression (SQLSTATE 42822), or any function that is variant or has an external action (SQLSTATE 42845).

More complex forms of the GROUP BY clause include *grouping-sets* and *super-groups*. For a description of these forms, see "grouping-sets" on page 1227 and "super-groups" on page 1228, respectively.

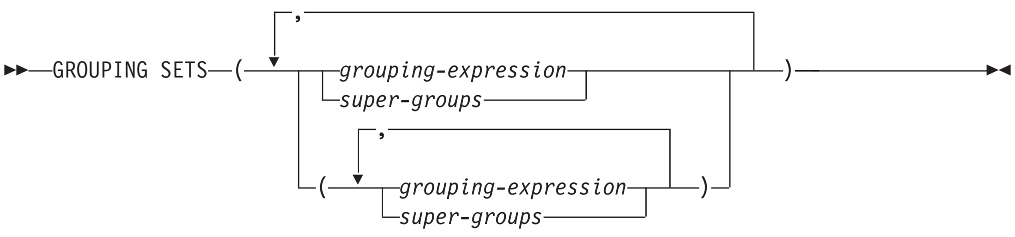
The result of GROUP BY is a set of groups of rows. Each row in this result represents the set of rows for which the *grouping-expression* is equal. For grouping, all null values from a *grouping-expression* are considered equal.

A *grouping-expression* can be used in a search condition in a HAVING clause, in an expression in a SELECT clause or in a *sort-key-expression* of an ORDER BY clause (see "order-by-clause" on page 1232 for details). In each case, the reference specifies only one value for each group. For example, if the *grouping-expression* is *col1+col2*, then an allowed expression in the select list would be *col1+col2+3*. Associativity rules for expressions would disallow the similar expression, *3+col1+col2*, unless parentheses are used to ensure that the corresponding expression is evaluated in the same order. Thus, *3+(col1+col2)* would also be allowed in the select list. If the concatenation operator is used, the *grouping-expression* must be used exactly as the expression was specified in the select list.

If the *grouping-expression* contains varying-length strings with trailing blanks, the values in the group can differ in the number of trailing blanks and may not all have the same length. In that case, a reference to the *grouping-expression* still specifies only one value for each group, but the value for a group is chosen arbitrarily from the available set of values. Thus, the actual length of the result value is unpredictable.

As noted, there are some cases where the GROUP BY clause cannot refer directly to a column that is specified in the SELECT clause as an expression (scalar-fullselect, variant or external action functions). To group using such an expression, use a nested table expression or a common table expression to first provide a result table with the expression as a column of the result. For an example using nested table expressions, see "Example A9" on page 1237.

grouping-sets



A *grouping-sets* specification allows multiple grouping clauses to be specified in a single statement. This can be thought of as the union of two or more groups of rows into a single result set. It is logically equivalent to the union of multiple subselects with the group by clause in each subselect corresponding to one grouping set. A grouping set can be a single element or can be a list of elements delimited by parentheses, where an element is either a grouping-expression or a super-group. Using *grouping-sets* allows the groups to be computed with a single pass over the base table.

The *grouping-sets* specification allows either a simple *grouping-expression* to be used, or the more complex forms of *super-groups*. For a description of *super-groups*, see "super-groups" on page 1228.

Note that grouping sets are the fundamental building blocks for GROUP BY operations. A simple GROUP BY with a single column can be considered a grouping set with one element. For example:

```
GROUP BY a
```

is the same as

```
GROUP BY GROUPING SETS((a))
```

and

```
GROUP BY a,b,c
```

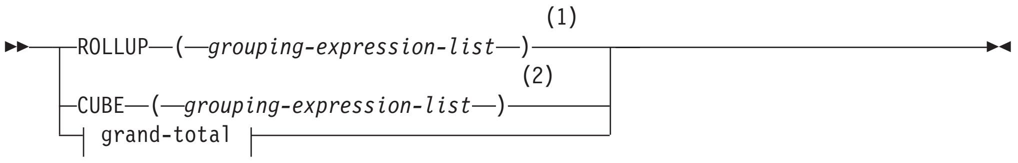
is the same as

```
GROUP BY GROUPING SETS((a,b,c))
```

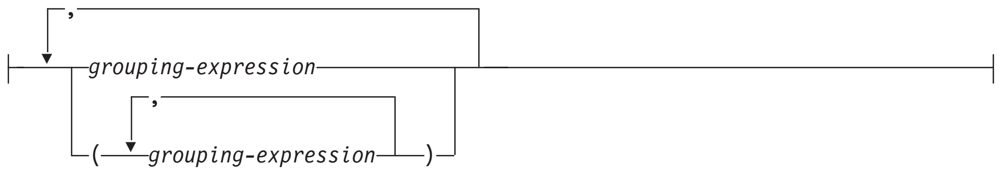
Non-aggregation columns from the select list of the subselect that are excluded from a grouping set will return a null for such columns for each row generated for that grouping set. This reflects the fact that aggregation was done without considering the values for those columns.

"Example C2" on page 1241 through "Example C7" on page 1244 illustrate the use of grouping sets.

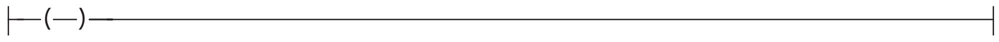
super-groups



grouping-expression-list:



grand-total:



Notes:

1 Alternate specification when used alone in group-by-clause is: grouping-expression-list WITH ROLLUP.

2 Alternate specification when used alone in group-by-clause is: grouping-expression-list WITH CUBE.

ROLLUP (*grouping-expression-list* **)**

A *ROLLUP grouping* is an extension to the GROUP BY clause that produces a result set containing *sub-total* rows in addition to the "regular" grouped rows. *Sub-total* rows are "super-aggregate" rows that contain further aggregates whose values are derived by applying the same column functions that were used to obtain the grouped rows. These rows are called sub-total rows, because that is their most common use; however, any column function can be used for the aggregation. For instance, MAX and AVG are used in "Example C8" on page 1246.

A ROLLUP grouping is a series of *grouping-sets*. The general specification of a ROLLUP with *n* elements

```
GROUP BY ROLLUP(C1,C2,...,Cn-1,Cn)
```

is equivalent to

```
GROUP BY GROUPING SETS((C1,C2,...,Cn-1,Cn)
                       (C1,C2,...,Cn-1)
                       ...
                       (C1,C2)
                       (C1)
                       () )
```

Note that the *n* elements of the ROLLUP translate to *n*+1 grouping sets. Note also that the order in which the *grouping-expressions* is specified is significant for ROLLUP. For example:

```
GROUP BY ROLLUP(a,b)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b)
                       (a)
                       () )
```

while

```
GROUP BY ROLLUP(b,a)
```

is the same as

```
GROUP BY GROUPING SETS((b,a)
                       (b)
                       () )
```

The ORDER BY clause is the only way to guarantee the order of the rows in the result set. “Example C3” on page 1241 illustrates the use of ROLLUP.

CUBE (*grouping-expression-list*)
A *CUBE grouping* is an extension to the GROUP BY clause that produces a result set that contains all the rows of a ROLLUP aggregation and, in addition, contains ″cross-tabulation″ rows. *Cross-tabulation* rows are additional ″super-aggregate″ rows that are not part of an aggregation with sub-totals.

Like a ROLLUP, a CUBE grouping can also be thought of as a series of *grouping-sets*. In the case of a CUBE, all permutations of the cubed *grouping-expression-list* are computed along with the grand total. Therefore, the *n* elements of a CUBE translate to 2***n* (2 to the power *n*) *grouping-sets*. For instance, a specification of

```
GROUP BY CUBE(a,b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a,c)
                       (b,c)
                       (a)
                       (b)
                       (c)
                       () )
```

Notice that the 3 elements of the CUBE translate to 8 grouping sets.

The order of specification of elements does not matter for CUBE. ’CUBE (DayOfYear, Sales_Person)’ and ’CUBE (Sales_Person, DayOfYear)’ yield the same result sets. The use of the word ’same’ applies to content of the result set, not to its order. The ORDER BY clause is the only way to guarantee the order of the rows in the result set. “Example C4” on page 1241 illustrates the use of CUBE.

grouping-expression-list
A *grouping-expression-list* is used within a CUBE or ROLLUP clause to define the number of elements in the CUBE or ROLLUP operation. This is controlled by using parentheses to delimit elements with multiple *grouping-expression*s.

The rules for a *grouping-expression* are described in “group-by-clause” on page 1226. For example, suppose that a query is to return the total expenses for the ROLLUP of City within a Province but not within a County. However the clause:

```
GROUP BY ROLLUP(Province, County, City)
```

results in unwanted sub-total rows for the County. In the clause

```
GROUP BY ROLLUP(Province, (County, City))
```

the composite (County, City) forms one element in the ROLLUP and, therefore, a query that uses this clause will yield the desired result. In other words, the two element ROLLUP

```
GROUP BY ROLLUP(Province, (County, City))
```

generates

```
GROUP BY GROUPING SETS((Province, County, City)
                       (Province)
                       () )
```

while the 3 element ROLLUP would generate

```
GROUP BY GROUPING SETS((Province, County, City)
                       (Province, County)
                       (Province)
                       () )
```

“Example C2” on page 1241 also utilizes composite column values.

grand-total
Both CUBE and ROLLUP return a row which is the overall (grand total) aggregation. This may be separately specified with empty parentheses within the GROUPING SET clause. It may also be specified directly in the GROUP BY clause, although there is no effect on the result of the query. “Example C4” on page 1241 uses the grand-total syntax.

Combining grouping sets

This can be used to combine any of the types of GROUP BY clauses. When simple *grouping-expression* fields are combined with other groups, they are ″appended″ to the beginning of the resulting *grouping sets*. When ROLLUP or CUBE expressions are combined, they operate like ″multipliers″ on the remaining expression, forming additional grouping set entries according to the definition of either ROLLUP or CUBE.

For instance, combining *grouping-expression* elements acts as follows:

```
GROUP BY a, ROLLUP(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a) )
```

Or similarly,

```
GROUP BY a, b, ROLLUP(c,d)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c,d)
                       (a,b,c)
                       (a,b) )
```

Combining of *ROLLUP* elements acts as follows:

```
GROUP BY ROLLUP(a), ROLLUP(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a)
                       (b,c)
                       (b)
                       () )
```

Similarly,

```
GROUP BY ROLLUP(a), CUBE(b,c)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c)
                       (a,b)
                       (a,c)
                       (a)
                       (b,c)
                       (b)
                       (c)
                       () )
```

Combining of *CUBE* and *ROLLUP* elements acts as follows:

```
GROUP BY CUBE(a,b), ROLLUP(c,d)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b,c,d)
                       (a,b,c)
                       (a,b)
                       (a,c,d)
                       (a,c)
                       (a)
                       (b,c,d)
                       (b,c)
                       (b)
                       (c,d)
                       (c)
                       () )
```

Like a simple *grouping-expression*, combining grouping sets also eliminates duplicates within each grouping set. For instance,

```
GROUP BY a, ROLLUP(a,b)
```

is equivalent to

```
GROUP BY GROUPING SETS((a,b)
                       (a) )
```

A more complete example of combining grouping sets is to construct a result set that eliminates certain rows that would be returned for a full CUBE aggregation.

For example, consider the following GROUP BY clause:

```
GROUP BY Region,
         ROLLUP(Sales_Person, WEEK(Sales_Date)),
         CUBE(YEAR(Sales_Date), MONTH (Sales_Date))
```

The column listed immediately to the right of GROUP BY is simply grouped, those within the parenthesis following ROLLUP are rolled up, and those within the parenthesis following CUBE are cubed. Thus, the above clause results in a cube of MONTH within YEAR which is then rolled up within WEEK within Sales_Person within the Region aggregation. It does not result in any grand total row or any cross-tabulation rows on Region, Sales_Person or WEEK(Sales_Date) so produces fewer rows than the clause:

```
GROUP BY ROLLUP (Region, Sales_Person, WEEK(Sales_Date),
                 YEAR(Sales_Date), MONTH(Sales_Date) )
```

having-clause

```
►►──HAVING──search-condition──────────────────────────────────────────►◄
```

The HAVING clause specifies an intermediate result table that consists of those groups of R for which the *search-condition* is true. R is the result of the previous clause of the subselect. If this clause is not GROUP BY, R is considered to be a single group with no grouping columns.

Each *column-name* in the search condition must do one of the following:

- Unambiguously identify a grouping column of R.
- Be specified within a column function.
- Be a correlated reference. A *column-name* is a correlated reference if it identifies a column of a *table-reference* in an outer subselect.

A group of R to which the search condition is applied supplies the argument for each column function in the search condition, except for any function whose argument is a correlated reference.

If the search condition contains a subquery, the subquery can be thought of as being executed each time the search condition is applied to a group of R, and the results used in applying the search condition. In actuality, the subquery is executed for each group only if it contains a correlated reference. For an illustration of the difference, see “Example A6” on page 1236 and “Example A7” on page 1236.

A correlated reference to a group of R must either identify a grouping column or be contained within a column function.

When HAVING is used without GROUP BY, the select list can only be a column name within a column function, a correlated column reference, a literal, or a special register.

order-by-clause

```
                 ┌─,──────────────────────────────┐
                 ▼                   ┌─ASC──┐     │
►►──ORDER BY──┬──┬─┤ sort-key ├──┼──────┼─┬─┴─┬──────────────────────►◄
              │  │                 └─DESC─┘ │   │
              │  └─ORDER OF──table-designator─┘   │
              └─INPUT SEQUENCE──────────────────┘
```

sort-key:

```
├──┬─simple-column-name───┬──────────────────────────────────────────┤
   ├─simple-integer───────┤
   └─sort-key-expression──┘
```

The ORDER BY clause specifies an ordering of the rows of the result table. If a single sort specification (one *sort-key* with associated direction) is identified, the rows are ordered by the values of that sort specification. If more than one sort specification is identified, the rows are ordered by the values of the first identified sort specification, then by the values of the second identified sort specification, and so on. Each *sort-key* cannot have a data type of LONG VARCHAR, CLOB, LONG VARGRAPHIC, DBCLOB, BLOB, DATALINK, distinct type on any of these types, or structured type (SQLSTATE 42907).

A named column in the select list may be identified by a *sort-key* that is a *simple-integer* or a *simple-column-name*. An unnamed column in the select list must be identified by an *simple-integer* or, in some cases, by a *sort-key-expression* that matches the expression in the select list (see details of *sort-key-expression*). A column is unnamed if the AS clause is not specified and it is derived from a constant, an expression with operators, or a function.

Ordering is performed in accordance with comparison rules. The null value is higher than all other values. If the ORDER BY clause does not completely order the rows, rows with duplicate values of all identified columns are displayed in an arbitrary order.

simple-column-name

Usually identifies a column of the result table. In this case, *simple-column-name* must be the column name of a named column in the select list.

The *simple-column-name* may also identify a column name of a table, view, or nested table identified in the FROM clause if the query is a subselect. An error occurs if the subselect:

- Specifies DISTINCT in the select-clause (SQLSTATE 42822)
- Produces a grouped result and the *simple-column-name* is not a *grouping-expression* (SQLSTATE 42803).

Determining which column is used for ordering the result is described under "Column names in sort keys" below.

simple-integer

Must be greater than 0 and not greater than the number of columns in the result table (SQLSTATE 42805). The integer *n* identifies the *n*th column of the result table.

sort-key-expression

An expression that is not simply a column name or an unsigned integer constant. The query to which ordering is applied must be a *subselect* to use this form of sort-key. The *sort-key-expression* cannot include a correlated scalar fullselect (SQLSTATE 42703), an XMLQUERY or XMLEXISTS expression (SQLSTATE 42822), or a function with an external action (SQLSTATE 42845).

Any column-name within a *sort-key-expression* must conform to the rules described under "Column names in sort keys" below.

There are a number of special cases that further restrict the expressions that can be specified.

- DISTINCT is specified in the SELECT clause of the subselect (SQLSTATE 42822).

  The sort-key-expression must match exactly with an expression in the select list of the subselect (scalar-fullselects are never matched).
- The subselect is grouped (SQLSTATE 42803).

The sort-key-expression can:

– be an expression in the select list of the subselect,
– include a *grouping-expression* from the GROUP BY clause of the subselect
– include a column function, constant or host variable.

ASC
Uses the values of the column in ascending order. This is the default.

DESC
Uses the values of the column in descending order.

ORDER OF *table-designator*
Specifies that the same ordering used in *table-designator* should be applied to the result table of the subselect. There must be a table reference matching *table-designator* in the FROM clause of the subselect that specifies this clause (SQLSTATE 42703). The subselect (or fullselect) corresponding to the specified *table-designator* must include an ORDER BY clause that is dependant on the data (SQLSTATE 428FI). The ordering that is applied is the same as if the columns of the ORDER BY clause in the nested subselect (or fullselect) were included in the outer subselect (or fullselect), and these columns were specified in place of the ORDER OF clause.

Note that this form is not allowed in a fullselect (other than the degenerative form of a fullselect). For example, the following is not valid:

```
(SELECT C1 FROM T1
   ORDER BY C1)
UNION
SELECT C1 FROM T2
   ORDER BY ORDER OF T1
```

The following example *is* valid:

```
SELECT C1 FROM
   (SELECT C1 FROM T1
      UNION
    SELECT C1 FROM T2
    ORDER BY C1 ) AS UTABLE
ORDER BY ORDER OF UTABLE
```

INPUT SEQUENCE
Specifies that, for an INSERT statement, the result table will reflect the input order of ordered data rows. INPUT SEQUENCE ordering can only be specified if an INSERT statement is used in a FROM clause (SQLSTATE 428G4). See "table-reference" on page 1216. If INPUT SEQUENCE is specified and the input data is not ordered, the INPUT SEQUENCE clause is ignored.

Notes:

- **Column names in sort keys**:
  – The column name is qualified.

    The query must be a *subselect* (SQLSTATE 42877). The column name must unambiguously identify a column of some table, view or nested table in the FROM clause of the subselect (SQLSTATE 42702). The value of the column is used to compute the value of the sort specification.
  – The column name is unqualified.
    - The query is a subselect.

      If the column name is identical to the name of more than one column of the result table, the column name must unambiguously identify a column of some table, view or nested table in the FROM clause of the ordering subselect (SQLSTATE 42702). If the column name is identical to one column, that column is used to compute the value of the sort specification. If the column name is not identical to a column of the result table, then it must unambiguously identify a column of some table, view or nested table in the FROM clause of the fullselect in the select-statement (SQLSTATE 42702).

- The query is not a subselect (it includes set operations such as union, except or intersect).

  The column name must not be identical to the name of more than one column of the result table (SQLSTATE 42702). The column name must be identical to exactly one column of the result table (SQLSTATE 42707), and this column is used to compute the value of the sort specification.

- **Limits**: The use of a *sort-key-expression* or a *simple-column-name* where the column is not in the select list may result in the addition of the column or expression to the temporary table used for sorting. This may result in reaching the limit of the number of columns in a table or the limit on the size of a row in a table. Exceeding these limits will result in an error if a temporary table is required to perform the sorting operation.

fetch-first-clause

```
                   ┌─1───────┐
►►──FETCH FIRST────┼─────────┼──┬─ROW──┬──ONLY───────────────────────────────►◄
                   └─integer─┘  └─ROWS─┘
```

The *fetch-first-clause* sets a maximum number of rows that can be retrieved. It lets the database manager know that the application does not want to retrieve more than *integer* rows, regardless of how many rows there might be in the result table when this clause is not specified. An attempt to fetch beyond *integer* rows is handled the same way as normal end of data (SQLSTATE 02000). The value of *integer* must be a positive integer (not zero).

Limiting the result table to the first *integer* rows can improve performance. The database manager will cease processing the query once it has determined the first *integer* rows. If both the *fetch-first-clause* and the *optimize-for-clause* are specified, the lower of the *integer* values from these clauses is used to influence the communications buffer size. The values are considered independently for optimization purposes.

If the fullselect contains an SQL data change statement in the FROM clause, all the rows are modified regardless of the limit on the number of rows to fetch.

Examples of subselects

Example A1: Select all columns and rows from the EMPLOYEE table.

```
SELECT * FROM EMPLOYEE
```

Example A2: Join the EMP_ACT and EMPLOYEE tables, select all the columns from the EMP_ACT table and add the employee's surname (LASTNAME) from the EMPLOYEE table to each row of the result.

```
 SELECT EMP_ACT.*, LASTNAME
   FROM EMP_ACT, EMPLOYEE
   WHERE EMP_ACT.EMPNO = EMPLOYEE.EMPNO
```

Example A3: Join the EMPLOYEE and DEPARTMENT tables, select the employee number (EMPNO), employee surname (LASTNAME), department number (WORKDEPT in the EMPLOYEE table and DEPTNO in the DEPARTMENT table) and department name (DEPTNAME) of all employees who were born (BIRTHDATE) earlier than 1930.

```
SELECT EMPNO, LASTNAME, WORKDEPT, DEPTNAME
   FROM EMPLOYEE, DEPARTMENT
   WHERE WORKDEPT = DEPTNO
   AND YEAR(BIRTHDATE) < 1930
```

Example A4: Select the job (JOB) and the minimum and maximum salaries (SALARY) for each group of rows with the same job code in the EMPLOYEE table, but only for groups with more than one row and with a maximum salary greater than or equal to 27000.

```
 SELECT JOB, MIN(SALARY), MAX(SALARY)
   FROM EMPLOYEE
   GROUP BY JOB
   HAVING COUNT(*) > 1
   AND MAX(SALARY) >= 27000
```

Example A5: Select all the rows of EMP_ACT table for employees (EMPNO) in department (WORKDEPT) 'E11'. (Employee department numbers are shown in the EMPLOYEE table.)

```
SELECT *
  FROM EMP_ACT
  WHERE EMPNO IN
           (SELECT EMPNO
               FROM EMPLOYEE
               WHERE WORKDEPT = 'E11')
```

Example A6: From the EMPLOYEE table, select the department number (WORKDEPT) and maximum departmental salary (SALARY) for all departments whose maximum salary is less than the average salary for all employees.

```
SELECT WORKDEPT, MAX(SALARY)
  FROM EMPLOYEE
  GROUP BY WORKDEPT
  HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                          FROM EMPLOYEE)
```

The subquery in the HAVING clause would only be executed once in this example.

Example A7: Using the EMPLOYEE table, select the department number (WORKDEPT) and maximum departmental salary (SALARY) for all departments whose maximum salary is less than the average salary in all other departments.

```
SELECT WORKDEPT, MAX(SALARY)
   FROM EMPLOYEE EMP_COR
    GROUP BY WORKDEPT
    HAVING MAX(SALARY) < (SELECT AVG(SALARY)
                          FROM EMPLOYEE
                          WHERE NOT WORKDEPT = EMP_COR.WORKDEPT)
```

In contrast to "Example A6," the subquery in the HAVING clause would need to be executed for each group.

Example A8: Determine the employee number and salary of sales representatives along with the average salary and head count of their departments.

This query must first create a nested table expression (DINFO) in order to get the AVGSALARY and EMPCOUNT columns, as well as the DEPTNO column that is used in the WHERE clause.

```
SELECT THIS_EMP.EMPNO, THIS_EMP.SALARY, DINFO.AVGSALARY, DINFO.EMPCOUNT
 FROM EMPLOYEE THIS_EMP,
       (SELECT OTHERS.WORKDEPT AS DEPTNO,
                          AVG(OTHERS.SALARY) AS AVGSALARY,
                          COUNT(*) AS EMPCOUNT
         FROM EMPLOYEE OTHERS
         GROUP BY OTHERS.WORKDEPT
        ) AS DINFO
 WHERE THIS_EMP.JOB = 'SALESREP'
  AND THIS_EMP.WORKDEPT = DINFO.DEPTNO
```

Using a nested table expression for this case saves the overhead of creating the DINFO view as a regular view. During statement preparation, accessing the catalog for the view is avoided and, because of the context of the rest of the query, only the rows for the department of the sales representatives need to be considered by the view.

Example A9: Display the average education level and salary for 5 random groups of employees.

This query requires the use of a nested table expression to set a random value for each employee so that it can subsequently be used in the GROUP BY clause.

```
SELECT RANDID , AVG(EDLEVEL), AVG(SALARY)
  FROM ( SELECT EDLEVEL, SALARY, INTEGER(RAND()*5) AS RANDID
           FROM EMPLOYEE
       ) AS EMPRAND
  GROUP BY RANDID
```

Example A10: Query the EMP_ACT table and return those project numbers that have an employee whose salary is in the top 10 of all employees.

```
SELECT EMP_ACT.EMPNO,PROJNO
   FROM EMP_ACT
   WHERE EMP_ACT.EMPNO IN
       (SELECT EMPLOYEE.EMPNO
        FROM EMPLOYEE
        ORDER BY SALARY DESC
        FETCH FIRST 10 ROWS ONLY)
```

Examples of joins

Example B1: This example illustrates the results of the various joins using tables J1 and J2. These tables contain rows as shown.

```
SELECT * FROM J1

W   X
--- ------
A       11
B       12
C       13

SELECT * FROM J2

Y   Z
--- ------
A       21
C       22
D       23
```

The following query does an inner join of J1 and J2 matching the first column of both tables.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
```

In this inner join example the row with column W='C' from J1 and the row with column Y='D' from J2 are not included in the result because they do not have a match in the other table. Note that the following alternative form of an inner join query produces the same result.

```
SELECT * FROM J1, J2 WHERE W=Y
```

The following left outer join will get back the missing row from J1 with nulls for the columns of J2. Every row from J1 is included.

```
SELECT * FROM J1 LEFT OUTER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
B       12 -        -
C       13 C       22
```

The following right outer join will get back the missing row from J2 with nulls for the columns of J1. Every row from J2 is included.

```
SELECT * FROM J1 RIGHT OUTER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
-        - D       23
```

The following full outer join will get back the missing rows from both J1 and J2 with nulls where appropriate. Every row from both J1 and J2 is included.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y

W   X      Y   Z
--- ------ --- ------
A       11 A       21
C       13 C       22
-        - D       23
B       12 -        -
```

Example B2: Using the tables J1 and J2 from the previous example, examine what happens when and additional predicate is added to the search condition.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=13

W   X      Y   Z
--- ------ --- ------
C       13 C       22
```

The additional condition caused the inner join to select only 1 row compared to the inner join in "Example B1" on page 1237.

Notice what the impact of this is on the full outer join.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=13

W   X      Y   Z
--- ------ --- ------
-        - A       21
C       13 C       22
-        - D       23
A       11 -        -
B       12 -        -
```

The result now has 5 rows (compared to 4 without the additional predicate) since there was only 1 row in the inner join and all rows of both tables must be returned.

The following query illustrates that placing the same additional predicate in WHERE clause has completely different results.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
   WHERE X=13

W   X      Y   Z
--- ------ --- ------
C       13 C       22
```

The WHERE clause is applied after the intermediate result of the full outer join. This intermediate result would be the same as the result of the full outer join query in “Example B1” on page 1237. The WHERE clause is applied to this intermediate result and eliminates all but the row that has X=13. Choosing the location of a predicate when performing outer joins can have significant impact on the results. Consider what happens if the predicate was X=12 instead of X=13. The following inner join returns no rows.

```
SELECT * FROM J1 INNER JOIN J2 ON W=Y AND X=12
```

Hence, the full outer join would return 6 rows, 3 from J1 with nulls for the columns of J2 and 3 from J2 with nulls for the columns of J1.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y AND X=12

W   X      Y   Z
--- ------ --- ------
-        - A       21
-        - C       22
-        - D       23
A       11 -        -
B       12 -        -
C       13 -        -
```

If the additional predicate is in the WHERE clause instead, 1 row is returned.

```
SELECT * FROM J1 FULL OUTER JOIN J2 ON W=Y
   WHERE X=12

W   X      Y   Z
--- ------ --- ------
B       12 -        -
```

Example B3: List every department with the employee number and last name of the manager, including departments without a manager.

```
SELECT DEPTNO, DEPTNAME, EMPNO, LASTNAME
  FROM DEPARTMENT LEFT OUTER JOIN EMPLOYEE
       ON MGRNO = EMPNO
```

Example B4: List every employee number and last name with the employee number and last name of their manager, including employees without a manager.

```
SELECT E.EMPNO, E.LASTNAME, M.EMPNO, M.LASTNAME
  FROM EMPLOYEE E LEFT OUTER JOIN
                                   DEPARTMENT INNER JOIN EMPLOYEE M
       ON MGRNO = M.EMPNO
       ON E.WORKDEPT = DEPTNO
```

The inner join determines the last name for any manager identified in the DEPARTMENT table and the left outer join guarantees that each employee is listed even if a corresponding department is not found in DEPARTMENT.

Examples of grouping sets, cube, and rollup

The queries in "Example C1" through "Example C4" on page 1241 use a subset of the rows in the SALES tables based on the predicate 'WEEK(SALES_DATE) = 13'.

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SALES AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
```

which results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 LUCCHESSI                 3
         13           6 LUCCHESSI                 1
         13           6 LEE                       2
         13           6 LEE                       2
         13           6 LEE                       3
         13           6 LEE                       5
         13           6 GOUNOT                    3
         13           6 GOUNOT                    1
         13           6 GOUNOT                    7
         13           7 LUCCHESSI                 1
         13           7 LUCCHESSI                 2
         13           7 LUCCHESSI                 1
         13           7 LEE                       7
         13           7 LEE                       3
         13           7 LEE                       7
         13           7 LEE                       4
         13           7 GOUNOT                    2
         13           7 GOUNOT                   18
         13           7 GOUNOT                    1
```

Example C1: Here is a query with a basic GROUP BY clause over 3 columns:

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 GOUNOT                   11
         13           6 LEE                      12
         13           6 LUCCHESSI                 4
         13          7 GOUNOT                   21
         13          7 LEE                      21
         13          7 LUCCHESSI                 4
```

Example C2: Produce the result based on two different grouping sets of rows from the SALES table.

```
  SELECT WEEK(SALES_DATE) AS WEEK,
         DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
         SALES_PERSON, SUM(SALES) AS UNITS_SOLD
  FROM SALES
  WHERE WEEK(SALES_DATE) = 13
  GROUP BY GROUPING SETS ( (WEEK(SALES_DATE), SALES_PERSON),
                           (DAYOFWEEK(SALES_DATE), SALES_PERSON))
  ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
  WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
  ----------- ----------- --------------- -----------
           13           - GOUNOT                   32
           13           - LEE                      33
           13           - LUCCHESSI                 8
            -           6 GOUNOT                   11
            -           6 LEE                      12
            -           6 LUCCHESSI                 4
            -           7 GOUNOT                   21
            -           7 LEE                      21
            -           7 LUCCHESSI                 4
```

The rows with WEEK 13 are from the first grouping set and the other rows are from the second grouping set.

Example C3: If you use the 3 distinct columns involved in the grouping sets of "Example C2" and perform a ROLLUP, you can see grouping sets for (WEEK,DAY_WEEK,SALES_PERSON), (WEEK, DAY_WEEK), (WEEK) and grand total.

```
SELECT WEEK(SALES_DATE) AS WEEK,
        DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
        SALES_PERSON, SUM(SALES) AS UNITS_SOLD
  FROM SALES
  WHERE WEEK(SALES_DATE) = 13
  GROUP BY ROLLUP ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON )
  ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
  WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
  ----------- ----------- --------------- -----------
           13           6 GOUNOT                   11
           13           6 LEE                      12
           13           6 LUCCHESSI                 4
           13           6 -                        27
           13           7 GOUNOT                   21
           13           7 LEE                      21
           13           7 LUCCHESSI                 4
           13           7 -                        46
           13           - -                        73
            -           - -                        73
```

Example C4: If you run the same query as "Example C3" only replace ROLLUP with CUBE, you can see additional grouping sets for (WEEK,SALES_PERSON), (DAY_WEEK,SALES_PERSON), (DAY_WEEK), (SALES_PERSON) in the result.

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SALES_PERSON, SUM(SALES) AS UNITS_SOLD
FROM SALES
WHERE WEEK(SALES_DATE) = 13
GROUP BY CUBE ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE), SALES_PERSON )
ORDER BY WEEK, DAY_WEEK, SALES_PERSON
```

This results in:

```
WEEK        DAY_WEEK    SALES_PERSON    UNITS_SOLD
----------- ----------- --------------- -----------
         13           6 GOUNOT                   11
         13           6 LEE                      12
         13           6 LUCCHESSI                 4
         13           6 -                        27
         13           7 GOUNOT                   21
         13           7 LEE                      21
         13           7 LUCCHESSI                 4
         13           7 -                        46
         13           - GOUNOT                   32
         13           - LEE                      33
         13           - LUCCHESSI                 8
         13           - -                        73
          -           6 GOUNOT                   11
          -           6 LEE                      12
          -           6 LUCCHESSI                 4
          -           6 -                        27
          -           7 GOUNOT                   21
          -           7 LEE                      21
          -           7 LUCCHESSI                 4
          -           7 -                        46
          -           - GOUNOT                   32
          -           - LEE                      33
          -           - LUCCHESSI                 8
          -           - -                        73
```

Example C5: Obtain a result set which includes a grand-total of selected rows from the SALES table together with a group of rows aggregated by SALES_PERSON and MONTH.

```
SELECT SALES_PERSON,
       MONTH(SALES_DATE) AS MONTH,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY GROUPING SETS ( (SALES_PERSON, MONTH(SALES_DATE)),
                         ()
                       )
ORDER BY SALES_PERSON, MONTH
```

This results in:

```
SALES_PERSON    MONTH       UNITS_SOLD
--------------- ----------- -----------
GOUNOT                    3          35
GOUNOT                    4          14
GOUNOT                   12           1
LEE                       3          60
LEE                       4          25
LEE                      12           6
LUCCHESSI                 3           9
LUCCHESSI                 4           4
LUCCHESSI                12           1
-                         -         155
```

Example C6: This example shows two simple ROLLUP queries followed by a query which treats the two ROLLUPs as grouping sets in a single result set and specifies row ordering for each column involved in the grouping sets.

Example C6-1:

```
SELECT WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP ( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) )
ORDER BY WEEK, DAY_WEEK
```

results in:

```
WEEK        DAY_WEEK    UNITS_SOLD
----------- ----------- -----------
         13           6          27
         13           7          46
         13           -          73
         14           1          31
         14           2          43
         14           -          74
         53           1           8
         53           -           8
          -           -         155
```

Example C6-2:

```
SELECT MONTH(SALES_DATE) AS MONTH,
       REGION,
       SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP ( MONTH(SALES_DATE), REGION );
ORDER BY MONTH, REGION
```

results in:

```
MONTH       REGION          UNITS_SOLD
----------- --------------- -----------
          3 Manitoba                 22
          3 Ontario-North             8
          3 Ontario-South            34
          3 Quebec                   40
          3 -                       104
          4 Manitoba                 17
          4 Ontario-North             1
          4 Ontario-South            14
          4 Quebec                   11
          4 -                        43
         12 Manitoba                  2
         12 Ontario-South             4
         12 Quebec                    2
         12 -                         8
          - -                       155
```

Example C6-3:

```
  SELECT WEEK(SALES_DATE) AS WEEK,
         DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
         MONTH(SALES_DATE) AS MONTH,
         REGION,
         SUM(SALES) AS UNITS_SOLD
  FROM SALES
  GROUP BY GROUPING SETS ( ROLLUP( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) ),
                           ROLLUP( MONTH(SALES_DATE), REGION )  )
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

results in:

```
WEEK        DAY_WEEK    MONTH       REGION          UNITS_SOLD
----------- ----------- ----------- --------------- -----------
         13           6           - -                        27
         13           7           - -                        46
         13           -           - -                        73
         14           1           - -                        31
         14           2           - -                        43
         14           -           - -                        74
         53           1           - -                         8
         53           -           - -                         8
          -           -           3 Manitoba                 22
          -           -           3 Ontario-North             8
          -           -           3 Ontario-South            34
          -           -           3 Quebec                   40
          -           -           3 -                       104
          -           -           4 Manitoba                 17
          -           -           4 Ontario-North             1
          -           -           4 Ontario-South            14
          -           -           4 Quebec                   11
          -           -           4 -                        43
          -           -          12 Manitoba                  2
          -           -          12 Ontario-South             4
          -           -          12 Quebec                    2
          -           -          12 -                         8
          -           -           - -                       155
          -           -           - -                       155
```

Using the two ROLLUPs as grouping sets causes the result to include duplicate rows. There are even two grand total rows.

Observe how the use of ORDER BY has affected the results:
- In the first grouped set, week 53 has been repositioned to the end.
- In the second grouped set, month 12 has now been positioned to the end and the regions now appear in alphabetic order.
- Null values are sorted high.

Example C7: In queries that perform multiple ROLLUPs in a single pass (such as “Example C6-3” on page 1243) you may want to be able to indicate which grouping set produced each row. The following steps demonstrate how to provide a column (called GROUP) which indicates the origin of each row in the result set. By origin, we mean which one of the two grouping sets produced the row in the result set.

Step 1: Introduce a way of ″generating″ new data values, using a query which selects from a VALUES clause (which is an alternate form of a fullselect). This query shows how a table can be derived called ″X″ having 2 columns ″R1″ and ″R2″ and 1 row of data.

```
SELECT R1,R2
FROM (VALUES('GROUP 1','GROUP 2')) AS X(R1,R2);
```

results in:

```
R1      R2
------- -------
GROUP 1 GROUP 2
```

Step 2: Form the cross product of this table ″X″ with the SALES table. This add columns ″R1″ and ″R2″ to every row.

```
SELECT R1, R2, WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION,
       SALES AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
```

This add columns ″R1″ and ″R2″ to every row.

Step 3: Now we can combine these columns with the grouping sets to include these columns in the rollup analysis.

```
SELECT R1, R2,
       WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE),
                           DAYOFWEEK(SALES_DATE))),
                         (R2,ROLLUP( MONTH(SALES_DATE), REGION ) )  )
ORDER BY WEEK, DAY_WEEK, MONTH, REGION
```

results in:

```
R1      R2      WEEK     DAY_WEEK  MONTH     REGION          UNITS_SOLD
------- ------- -------- --------- --------- --------------- -----------
GROUP 1 -             13         6         - -                        27
GROUP 1 -             13         7         - -                        46
GROUP 1 -             13         -         - -                        73
GROUP 1 -             14         1         - -                        31
GROUP 1 -             14         2         - -                        43
GROUP 1 -             14         -         - -                        74
GROUP 1 -             53         1         - -                         8
GROUP 1 -             53         -         - -                         8
-       GROUP 2        -         -         3 Manitoba                 22
-       GROUP 2        -         -         3 Ontario-North             8
-       GROUP 2        -         -         3 Ontario-South            34
-       GROUP 2        -         -         3 Quebec                   40
-       GROUP 2        -         -         3 -                       104
-       GROUP 2        -         -         4 Manitoba                 17
-       GROUP 2        -         -         4 Ontario-North             1
-       GROUP 2        -         -         4 Ontario-South            14
-       GROUP 2        -         -         4 Quebec                   11
-       GROUP 2        -         -         4 -                        43
-       GROUP 2        -         -        12 Manitoba                  2
-       GROUP 2        -         -        12 Ontario-South             4
-       GROUP 2        -         -        12 Quebec                    2
-       GROUP 2        -         -        12 -                         8
-       GROUP 2        -         -         - -                       155
GROUP 1 -              -         -         - -                       155
```

Step 4: Notice that because R1 and R2 are used in different grouping sets, whenever R1 is non-null in the result, R2 is null and whenever R2 is non-null in the result, R1 is null. That means you can consolidate these columns into a single column using the COALESCE function. You can also use this column in the ORDER BY clause to keep the results of the two grouping sets together.

```
SELECT COALESCE(R1,R2) AS GROUP,
       WEEK(SALES_DATE) AS WEEK,
       DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
       MONTH(SALES_DATE) AS MONTH,
       REGION, SUM(SALES) AS UNITS_SOLD
FROM SALES,(VALUES('GROUP 1','GROUP 2')) AS X(R1,R2)
GROUP BY GROUPING SETS ((R1, ROLLUP(WEEK(SALES_DATE),
                          DAYOFWEEK(SALES_DATE))),
                        (R2,ROLLUP( MONTH(SALES_DATE), REGION ) )  )
ORDER BY GROUP, WEEK, DAY_WEEK, MONTH, REGION;
```

results in:

```
  GROUP   WEEK        DAY_WEEK    MONTH       REGION          UNITS_SOLD
  ------- ----------- ----------- ----------- --------------- -----------
  GROUP 1          13           6           - -                        27
  GROUP 1          13           7           - -                        46
  GROUP 1          13           -           - -                        73
  GROUP 1          14           1           - -                        31
  GROUP 1          14           2           - -                        43
  GROUP 1          14           -           - -                        74
  GROUP 1          53           1           - -                         8
  GROUP 1          53           -           - -                         8
  GROUP 1           -           -           - -                       155
  GROUP 2           -           -           3 Manitoba                 22
  GROUP 2           -           -           3 Ontario-North             8
  GROUP 2           -           -           3 Ontario-South            34
  GROUP 2           -           -           3 Quebec                   40
  GROUP 2           -           -           3 -                       104
  GROUP 2           -           -           4 Manitoba                 17
  GROUP 2           -           -           4 Ontario-North             1
  GROUP 2           -           -           4 Ontario-South            14
  GROUP 2           -           -           4 Quebec                   11
  GROUP 2           -           -           4 -                        43
  GROUP 2           -           -          12 Manitoba                  2
  GROUP 2           -           -          12 Ontario-South             4
  GROUP 2           -           -          12 Quebec                    2
  GROUP 2           -           -          12 -                         8
  GROUP 2           -           -           - -                       155
```

Example C8: The following example illustrates the use of various column functions when performing a CUBE. The example also makes use of cast functions and rounding to produce a decimal result with reasonable precision and scale.

```
  SELECT MONTH(SALES_DATE) AS MONTH,
         REGION,
         SUM(SALES) AS UNITS_SOLD,
         MAX(SALES) AS BEST_SALE,
         CAST(ROUND(AVG(DECIMAL(SALES)),2) AS DECIMAL(5,2)) AS AVG_UNITS_SOLD
  FROM SALES
  GROUP BY CUBE(MONTH(SALES_DATE),REGION)
  ORDER BY MONTH, REGION
```

This results in:

```
MONTH       REGION          UNITS_SOLD  BEST_SALE   AVG_UNITS_SOLD
----------- --------------- ----------- ----------- --------------
          3 Manitoba                 22           7           3.14
          3 Ontario-North             8           3           2.67
          3 Ontario-South            34          14           4.25
          3 Quebec                   40          18           5.00
          3 -                       104          18           4.00
          4 Manitoba                 17           9           5.67
          4 Ontario-North             1           1           1.00
          4 Ontario-South            14           8           4.67
          4 Quebec                   11           8           5.50
          4 -                        43           9           4.78
         12 Manitoba                  2           2           2.00
         12 Ontario-South             4           3           2.00
         12 Quebec                    2           1           1.00
         12 -                         8           3           1.60
          - Manitoba                 41           9           3.73
          - Ontario-North             9           3           2.25
- Ontario-South            52          14            4.00
- Quebec                   53          18            4.42
- -                       155          18            3.87
```

Related reference:
- "Assignments and comparisons" in *SQL Reference, Volume 1*
- "Character strings" in *SQL Reference, Volume 1*
- "Fullselect" on page 1174
- "Functions" in *SQL Reference, Volume 1*
- "GROUPING aggregate function" in *SQL Reference, Volume 1*
- "Identifiers" on page 1372
- "Predicates" in *SQL Reference, Volume 1*
- "Select-statement" on page 1182
- "XMLTABLE table function" in *SQL Reference, Volume 1*
- "CREATE FUNCTION (External Table) statement" in *SQL Reference, Volume 2*
- "CREATE FUNCTION (SQL Scalar, Table, or Row) statement" in *SQL Reference, Volume 2*
- "DELETE " on page 1047
- "INSERT " on page 1109
- "UPDATE " on page 1145

TRANSFER OWNERSHIP

The TRANSFER OWNERSHIP statement transfers ownership of a database object.

Invocation:

This statement can be embedded in an application program or issued through the use of dynamic SQL statements. It is an executable statement that can be dynamically prepared only if DYNAMICRULES run behavior is in effect for the package (SQLSTATE 42509).

Authorization:

The privileges held by the authorization ID of the statement must include at least one of the following:
- Ownership of the object
- SECADM authority

Syntax:

```
►►──TRANSFER OWNERSHIP OF──┤ objects ├──TO──┤ new-owner ├──PRESERVE PRIVILEGES─────►◄
```

objects:

```
├──┬─ALIAS──alias-name──────────────────────────────────────────────────┬──────┤
   ├─CONSTRAINT──table-name.constraint-name────────────────────────────┤
   ├─DATABASE PARTITION GROUP──db-partition-group-name─────────────────┤
   ├─EVENT MONITOR──event-monitor-name─────────────────────────────────┤
   ├─FUNCTION──function-name──┬───────────────────────┬────────────────┤
   │                          └─(──┬─────────────┬──)─┘                │
   │                               │ ┌─,───────┐ │                     │
   │                               └─▼─data-type─┴─┘                   │
   ├─SPECIFIC FUNCTION──specific-name──────────────────────────────────┤
   ├─FUNCTION MAPPING──function-mapping-name───────────────────────────┤
   ├─INDEX──index-name─────────────────────────────────────────────────┤
   ├─INDEX EXTENSION──index-extension-name─────────────────────────────┤
   ├─METHOD──method-name──┬───────────────────────┬──FOR──type-name────┤
   │                      └─(──┬─────────────┬──)─┘                    │
   │                           │ ┌─,───────┐ │                         │
   │                           └─▼─data-type─┴─┘                       │
   ├─SPECIFIC METHOD──specific-name────────────────────────────────────┤
   ├─NICKNAME──nickname────────────────────────────────────────────────┤
   ├─PACKAGE──┬───────────────┬──package-id──┬──────────────────────┬──┤
   │          └─schema-name.──┘              └─┬─────────┬─version-id─┘ │
   │                                           └─VERSION─┘              │
   ├─PROCEDURE──procedure-name──┬───────────────────────┬──────────────┤
   │                            └─(──┬─────────────┬──)─┘              │
   │                                 │ ┌─,───────┐ │                   │
   │                                 └─▼─data-type─┴─┘                 │
   ├─SPECIFIC PROCEDURE──specific-name─────────────────────────────────┤
   ├─SCHEMA──schema-name───────────────────────────────────────────────┤
   ├─SEQUENCE──sequence-name───────────────────────────────────────────┤
   ├─TABLE──table-name─────────────────────────────────────────────────┤
   ├─TABLE HIERARCHY──root-table-name──────────────────────────────────┤
   ├─TABLESPACE──tablespace-name───────────────────────────────────────┤
   ├─TRIGGER──trigger-name─────────────────────────────────────────────┤
   ├─┬────────────┬──TYPE──type-name───────────────────────────────────┤
   │ └─DISTINCT───┘                                                    │
   ├─TYPE MAPPING──type-mapping-name───────────────────────────────────┤
   ├─VIEW──view-name───────────────────────────────────────────────────┤
   ├─VIEW HIERARCHY──root-view-name────────────────────────────────────┤
   └─XSROBJECT──xsrobject-name─────────────────────────────────────────┘
```

new-owner:

```
├──┬─USER──authorization-name──┬──────────────────────────────────────────┤
   ├─SESSION_USER──────────────┤
   └─SYSTEM_USER───────────────┘
```

Description:

ALIAS *alias-name*
: Identifies the alias that is to have its ownership transferred. The *alias-name* must identify an alias that is described in the catalog (SQLSTATE 42704).

  When ownership of the alias is transferred, the value in the OWNER column for the alias in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.

CONSTRAINT *table-name.constraint-name*
: Identifies the constraint that is to have its ownership transferred. The

table-name.constraint-name combination must identify a constraint and the table that it constrains. The *constraint-name* must identify a constraint that is described in the catalog (SQLSTATE 42704).

When ownership of the constraint is transferred, the value in the OWNER column for the constraint in the SYSCAT.TABCONST catalog view is replaced with the authorization ID of the new owner.

- If the constraint is a FOREIGN KEY constraint, the OWNER column in the SYSCAT.REFERENCES catalog view is replaced with the authorization ID of the new owner.
- If the constraint is a PRIMARY KEY or UNIQUE constraint, the OWNER column in the SYSCAT.INDEXES catalog view for the index that was created implicitly for this constraint is replaced with the authorization ID of the new owner. If the index existed, and it is reused in this case, the owner of the index is not changed.

DATABASE PARTITION GROUP *db-partition-group-name*

Identifies the database partition group that is to have its ownership transferred. The *db-partition-group-name* must identify a database partition group that is described in the catalog (SQLSTATE 42704).

When ownership of the database partition group is transferred, the value in the OWNER column for the database partition group in the SYSCAT.DBPARTITIONGROUPS catalog view is replaced with the authorization ID of the new owner.

EVENT MONITOR *event-monitor-name*

Identifies the event monitor that is to have its ownership transferred. The *event-monitor-name* must identify an event monitor that is described in the catalog (SQLSTATE 42704).

When ownership of the event monitor is transferred, the value in the OWNER column for the event monitor in the SYSCAT.EVENTMONITORS catalog view is replaced with the authorization ID of the new owner.

If the identified event monitor is active, an error is returned (SQLSTATE 429BT).

If there are event files in the target path of a WRITE TO FILE event monitor whose ownership is being transferred, the event files are not deleted.

When ownership of WRITE TO TABLE event monitors is transferred, table information in the SYSCAT.EVENTTABLES catalog view is retained.

FUNCTION

Identifies the function that is to have its ownership transferred. The specified function instance must be a user-defined function or function template that is described in the catalog. Ownership of functions that are implicitly generated by the CREATE DISTINCT TYPE and CREATE TYPE (Structured) statements cannot be transferred (SQLSTATE 429BT).

There are several different ways to identify the function instance.

FUNCTION *function-name*

Identifies the particular function that is to have its ownership transferred, and is valid only if there is exactly one function instance with that *function-name*. The function thus identified can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the function in the named or implied schema, an error is returned (SQLSTATE 42725).

FUNCTION *function-name* **(***data-type***,...)**
Provides the function signature, which uniquely identifies the function whose ownership is to be transferred. The function selection algorithm is not used.

function-name
Specifies the name of the function whose ownership is to be transferred. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Specified data types must match the types and positions that were specified on the CREATE FUNCTION statement. The number of data types and the logical concatenation of the data types are used to identify the specific function whose ownership is to be transferred.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because 0<*n*<25 means REAL, and 24<*n*<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

The FOR BIT DATA attribute is not considered to be part of the signature for matching purposes. So, for example, a CHAR FOR BIT DATA specified in the signature would match a function defined with CHAR only; the reverse would also be true.

If no function with the specified signature exists in the named or implied schema, an error is returned (SQLSTATE 42883).

When ownership of the function is transferred, the value in the OWNER column for the function in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

SPECIFIC FUNCTION *specific-name*
Identifies the particular user-defined function that is to have its ownership transferred, using the specific name either specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the

qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific function is transferred, the value in the OWNER column for the specific function in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

FUNCTION MAPPING *function-mapping-name*
Identifies the function mapping that is to have its ownership transferred. The *function-mapping-name* must identify a function mapping that is described in the catalog (SQLSTATE 42704).

When ownership of the function mapping is transferred, the value in the OWNER column for the function mapping in the SYSCAT.FUNCMAPPINGS catalog view is replaced with the authorization ID of the new owner.

INDEX *index-name*
Identifies the index or index specification that is to have its ownership transferred. The *index-name* must identify an index or index specification that is described in the catalog (SQLSTATE 42704).

When ownership of the index is transferred, the value in the OWNER column for the index in the SYSCAT.INDEXES catalog view is replaced with the authorization ID of the new owner.

Ownership of an index cannot be transferred if the table on which the index is defined is a global temporary table (SQLSTATE 429BT).

INDEX EXTENSION *index-extension-name*
Identifies the index extension that is to have its ownership transferred. The *index-extension-name* must identify an index extension that is described in the catalog (SQLSTATE 42704).

When ownership of the index extension is transferred, the value in the OWNER column for the index extension in the SYSCAT.INDEXEXTENSIONS catalog view is replaced with the authorization ID of the new owner.

METHOD
Identifies the method that is to have its ownership transferred. The method body specified must be a method that is described in the catalog (SQLSTATE 42704). The ownership of methods that are implicitly generated by the CREATE TYPE statement cannot be transferred (SQLSTATE 429BT).

There are several different ways to identify the method body.

METHOD *method-name*
Identifies the particular method that is to have its ownership transferred, and is valid only if there is exactly one method instance with name *method-name* and subject type *type-name*. Thus, the method identified can have any number of parameters. If no method by this name exists for the type *type-name*, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the method for the named data type, an error is returned (SQLSTATE 42725).

METHOD *method-name* **(***data-type***,...)**
Provides the method signature, which uniquely identifies the method whose ownership is to be transferred. The method selection algorithm is not used.

method-name
Specifies the name of the method whose ownership is to be transferred. The name must be an unqualified identifier.

(*data-type***, ...)**
Specified data types must match the types and positions that were specified on the CREATE TYPE or ALTER TYPE statement. The number of data types and the logical concatenation of the data types are used to identify the specific method instance whose ownership is to be transferred.

If *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because 0<*n*<25 means REAL and 24<*n*<54 means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the named data type, an error is returned (SQLSTATE 42883).

FOR *type-name*
Names the type for which the specified method is to have its ownership transferred. The name must identify a type that is described in the catalog (SQLSTATE 42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified type name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified type names.

When ownership of the method is transferred, the value in the OWNER column for the method in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

SPECIFIC METHOD *specific-name*
Identifies the particular method that is to have its ownership transferred. If the specific name is unqualified, the CURRENT SCHEMA special register is used as a qualifier for an unqualified specific name in dynamic SQL. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for an unqualified specific name. The *specific-name* must identify a method; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific method is transferred, the value in the OWNER column for the specific method in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

NICKNAME *nickname*
Identifies the nickname that is to have its ownership transferred. The *nickname* must be a nickname that is described in the catalog (SQLSTATE 42704).

When ownership of the nickname is transferred, the value in the OWNER column for the nickname in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.

PACKAGE *schema-name.package-id*
Identifies the package that is to have its ownership transferred. If a schema name is not specified, the package identifier is implicitly qualified by the default schema. The schema name and package identifier, together with the implicitly or explicitly specified version identifier, must identify a package that is described in the catalog (SQLSTATE 42704).

VERSION *version-id*
Identifies which package version is to have its ownership transferred. If a value is not specified, the version defaults to the empty string, and the ownership of this package is transferred. If multiple packages with the same package name but different versions exist, only the ownership of the package whose *version-id* is specified in the TRANSFER OWNERSHIP statement is transferred. Delimit the version identifier with double quotation marks when it:

- Is generated by the VERSION(AUTO) precompiler option
- Begins with a digit
- Contains lowercase or mixed-case letters

If the statement is invoked from an operating system command prompt, precede each double quotation mark delimiter with a back slash character to ensure that the operating system does not strip the delimiters.

When ownership of the package is transferred, the value in the BOUNDBY column for the package in the SYSCAT.PACKAGES catalog view is replaced with the authorization ID of the new owner.

The ownership of packages that are associated with SQL procedures cannot be transferred (SQLSTATE 429BT).

PROCEDURE
Identifies the procedure that is to have its ownership transferred. The procedure instance specified must be a procedure that is described in the catalog.

There are several different ways to identify the procedure instance.

PROCEDURE *procedure-name*
Identifies the particular procedure that is to have its ownership transferred, and is valid only if there is exactly one procedure with the *procedure-name* in the schema. The procedure thus identified can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error is returned (SQLSTATE 42704). If there is more than one specific instance of the procedure in the named or implied schema, an error is returned (SQLSTATE 42725).

PROCEDURE *procedure-name* **(***data-type***,...)**
Provides the procedure signature, which uniquely identifies the procedure whose ownership is to be transferred.

procedure-name
Specifies the procedure name of the procedure whose ownership is to be transferred. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
Specified data types must match the types and positions that were specified on the CREATE PROCEDURE statement. The number of data types and the logical concatenation of the data types are used to identify the specific procedure whose ownership is to be transferred.

If the *data-type* is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

However, if length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because $0<n<25$ means REAL and $24<n<54$ means DOUBLE. Matching occurs based on whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error is returned (SQLSTATE 42883).

When ownership of the procedure is transferred, the value in the OWNER column for the procedure in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

Transferring ownership of an SQL procedure that has an associated package also implicitly transfers ownership of the package to the new owner.

SPECIFIC PROCEDURE *specific-name*
Identifies the particular procedure that is to have its ownership transferred, using the specific name either specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error is returned (SQLSTATE 42704).

When ownership of the specific procedure is transferred, the value in the OWNER column for the specific procedure in the SYSCAT.ROUTINES catalog view is replaced with the authorization ID of the new owner.

SCHEMA *schema-name*
Identifies the schema that is to have its ownership transferred. The *schema-name* must identify a schema that is described in the catalog (SQLSTATE 42704).

When ownership of the schema is transferred, the value in the OWNER column for the schema in the SYSCAT.SCHEMATA catalog view is replaced with the authorization ID of the new owner.

SEQUENCE *sequence-name*
Identifies the sequence that is to have its ownership transferred. The *sequence-name* must identify a sequence that is described in the catalog (SQLSTATE 42704).

When ownership of the sequence is transferred, the value in the OWNER column for the schema in the SYSCAT.SEQUENCES catalog view is replaced with the authorization ID of the new owner.

TABLE *table-name*
Identifies the table that is to have its ownership transferred. The *table-name* must identify a table that exists in the database (SQLSTATE 42704) and must not identify a declared temporary table (SQLSTATE 42995).

When ownership of the table is transferred:

- The value in the OWNER column for the table in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the table in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

Ownership of subtables in a table hierarchy cannot be transferred (SQLSTATE 429BT).

In a federated system, ownership of a remote table that was created using transparent DDL can be transferred. Transferring the ownership of a remote table will not transfer ownership of the nickname that is associated with the table. Ownership of such a nickname can be transferred explicitly using the TRANSFER OWNERSHIP statement.

TABLE HIERARCHY *root-table-name*
Identifies the typed table that is the root table in a typed table hierarchy that is to have its ownership transferred. The *root-table-name* must identify a typed table that is the root table in the typed table hierarchy (SQLSTATE 428DR), and must refer to a typed table that exists in the database (SQLSTATE 42704).

When ownership of the table hierarchy is transferred:

- The value in the OWNER column for the root table and all of its subtables in the SYSCAT.TABLES catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the table and all of its subtables in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

TABLESPACE *tablespace-name*
Identifies the table space that is to have its ownership transferred. The *tablespace-name* must identify a table space that is described in the catalog (SQLSTATE 42704).

When ownership of the table space is transferred, the value in the OWNER column for the table space in the SYSCAT.TABLESPACES catalog view is replaced with the authorization ID of the new owner.

TRIGGER *trigger-name*
Identifies the trigger that is to have its ownership transferred. The *trigger-name* must identify a trigger that is described in the catalog (SQLSTATE 42704).

When ownership of the trigger is transferred, the value in the OWNER column for the trigger in the SYSCAT.TRIGGERS catalog view is replaced with the authorization ID of the new owner.

TYPE *type-name*
Identifies the user-defined type that is to have its ownership transferred. The *type-name* must identify a type that is described in the catalog (SQLSTATE 42704). If DISTINCT is specified, *type-name* must identify a distinct type that is described in the catalog (SQLSTATE 42704).

In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile or bind option implicitly specifies the qualifier for unqualified object names.

When ownership of the type is transferred, the value in the OWNER column for the type in the SYSCAT.DATATYPES catalog view is replaced with the authorization ID of the new owner.

TYPE MAPPING *type-mapping-name*
Identifies the user-defined data type mapping that is to have its ownership transferred. The *type-mapping-name* must identify a data type mapping that is described in the catalog (SQLSTATE 42704).

When ownership of the type mapping is transferred, the value in the OWNER column for the type mapping in the SYSCAT.TYPEMAPPINGS catalog view is replaced with the authorization ID of the new owner.

VIEW *view-name*
Identifies the view that is to have its ownership transferred. The *view-name* must identify a view that exists in the database (SQLSTATE 42704).

When ownership of the view is transferred:

- The value in the OWNER column for the view in the SYSCAT.VIEWS catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the view in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

The ownership of a subview in a view hierarchy cannot be transferred (SQLSTATE 429BT).

VIEW HIERARCHY *root-view-name*
Identifies the typed view that is the root view in a typed view hierarchy that is to have its ownership transferred. The *root-view-name* must identify a typed view that is the root view in the typed view hierarchy (SQLSTATE 428DR), and must refer to a typed view that exists in the database (SQLSTATE 42704).

When ownership of the view hierarchy is transferred:

- The value in the OWNER column for the root view and all of its subviews in the SYSCAT.VIEWS catalog view is replaced with the authorization ID of the new owner.
- The value in the OWNER column for all dependent objects on the view and all of its subviews in the SYSCAT.TABDEP catalog view is replaced with the authorization ID of the new owner.

XSROBJECT *xsrobject-name*
Identifies the XSR object that is to have its ownership transferred. The *xsrobject-name* must identify an XSR object that is described in the catalog (SQLSTATE 42704).

When ownership of the XSR object is transferred, the value in the OWNER column for the XSR object in the SYSCAT.XSROBJECTS catalog view is replaced with the authorization ID of the new owner.

USER *authorization-name*
Specifies the authorization ID to which ownership of the object is being transferred.

SESSION_USER
Specifies that the value of the SESSION_USER special register is to be used as the authorization ID to which ownership of the object is being transferred.

SYSTEM_USER
Specifies that the value of the SYSTEM_USER special register is to be used as the authorization ID to which ownership of the object is being transferred.

PRESERVE PRIVILEGES
Specifies that the current owner of an object that is to have its ownership transferred will continue to hold any existing privileges on the object after the transfer. For example, any privileges that were granted to the creator of a view when that view was created continue to be held by the original owner even after ownership has been transferred to another user.

Rules:

- Ownership of system-defined objects (where the owner is SYSIBM) cannot be transferred (SQLSTATE 42832).
- Ownership of schemas whose name starts with 'SYS' cannot be transferred (SQLSTATE 42832).
- Ownership of the following objects cannot be explicitly transferred (SQLSTATE 429BT):
  - Subtables in a table hierarchy (they are transferred with the root hierarchy table)
  - Subviews in a view hierarchy (they are transferred with the root hierarchy view)
  - Indexes that are defined on global temporary tables
  - Methods or functions that are implicitly generated when a user-defined type is created
  - Packages that depend on SQL procedures (they are transferred with the SQL procedure)
  - Event monitors that are active (they can be transferred when they are not active)
- An authorization ID that has SECADM authority cannot transfer the ownership of an object to itself, if it is not already the owner of the object (SQLSTATE 42502).

Notes:

- If the authorization ID to which the object is being transferred does not have sufficient privileges to maintain the object, ownership of the object will not be transferred, and an error is returned (SQLSTATE 42514).

  For example:

– Consider a view with SELECT and INSERT dependencies on an underlying table. The privileges held by the new owner of the view must include at least SELECT (with or without the GRANT OPTION) and INSERT (with or without the GRANT OPTION) for the ownership transfer to be successful. If the dependencies were SELECT WITH GRANT OPTION and INSERT WITH GRANT OPTION, the privileges held by the new owner of the view must include at least SELECT WITH GRANT OPTION and INSERT WITH GRANT OPTION.
– Consider a view with a dependency on a routine. The privileges held by the new owner of the view must include at least EXECUTE on the dependent routine.
– Consider a trigger with a dependency on a table. The privileges held by the new owner of the trigger must include the same set of privileges on the table that are indicated by the trigger's dependencies. ALTER privilege on the table on which the trigger is defined is not required.

The following table lists the system catalog views that describe the objects on which other database objects depend.

Table 146. Catalog Views that Describe Objects on which Other Objects Depend

Database Object	System Catalog View
CONSTRAINT	SYSCAT.CONSTDEP
FUNCTION	SYSCAT.ROUTINEDEP; SYSCAT.ROUTINES (for a sourced function)
INDEX	SYSCAT.INDEXDEP
INDEX EXTENSION	SYSCAT.INDEXEXTENSIONDEP
METHOD	SYSCAT.ROUTINEDEP
PACKAGE	SYSCAT.PACKAGEDEP
PROCEDURE	SYSCAT.ROUTINEDEP
TABLE	SYSCAT.TABDEP
TRIGGER	SYSCAT.TRIGDEP
VIEW	SYSCAT.TABDEP
XSROBJECT	SYSCAT.XSROBJECTDEP

If ownership of a database object that depends on another object is to be transferred successfully, the new owner of the database object must hold certain privileges on the dependent object of that dependency:
– If the dependent object is a sequence, the new owner must have the USAGE privilege on that sequence.
– If the dependent object is a function, method, or procedure, the new owner must have the EXECUTE privilege on that function, method, or procedure.
– If the dependent object is a package, the new owner must have the EXECUTE privilege on that package.
– If the dependent object is an XSR object, the new owner must have the USAGE privilege on that XSR object.

For any other dependent object of a dependency, use the TABAUTH column in the appropriate system catalog view to determine what privileges the new owner must hold.

- If an attempt is made to transfer ownership of an object to its owner, a warning is returned (SQLSTATE 01676).

- Ownership of the following database objects cannot be transferred, because these objects have no owner: buffer pools, servers, transformation functions, user mappings, and wrappers. Note that there is no OWNER column in the SYSCAT.BUFFERPOOLS, SYSCAT.SERVERS, SYSCAT.TRANSFORMS, SYSCAT.USEROPTIONS, and SYSCAT.WRAPPERS catalog tables.
- The schema name of an object whose ownership was transferred does not automatically change.
- If the current owner has had a privilege on the object revoked, and that privilege was subsequently granted back, the privilege is not transferred.
- ***Compatibilities***
  - For compatibility with previous versions of DB2:
    - NODEGROUP can be specified in place of DATABASE PARTITION GROUP

Examples:

Example 1: Transfer ownership of table T1 to PAUL.

```
TRANSFER OWNERSHIP OF TABLE WALID.T1
  TO USER PAUL PRESERVE PRIVILEGES
```

The value in the OWNER column for the table WALID.T1 in the SYSCAT.TABLES catalog view is replaced with ’PAUL’. Paul is implicitly granted the following privileges on table WALID.T1 (assuming that the previous owner of the table did not lose any privileges on it): CONTROL and ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, REFERENCE (WITH GRANT OPTION).

Example 2: Assume that JOHN creates tables T1 and T2, and that MIKE holds SELECT privilege on tables JOHN.T1 and JOHN.T2. MIKE creates view V1 that depends on tables JOHN.T1 and JOHN.T2. Transfer ownership of view V1 to HENRY, who has DBADM authority.

```
TRANSFER OWNERSHIP OF VIEW V1
  TO USER HENRY PRESERVE PRIVILEGES
```

The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with ’HENRY’. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = ’SYSIBM’, GRANTEE = ’HENRY’, and TABNAME = ’V1’.

Example 3: Assume that HENRY, who holds DBADM authority, creates a trigger TR1 that depends on table T1. Transfer ownership of trigger TR1 to WALID, who does not hold DBADM authority.

```
TRANSFER OWNERSHIP OF TRIGGER TR1
  TO USER WALID PRESERVE PRIVILEGES
```

Ownership of the trigger is transferred successfully, even though Walid does not hold DBADM authority.

Example 4: Assume that JOHN creates tables T1 and T2, and that MIKE holds SELECT privilege on table JOHN.T1 and CONTROL privilege on table JOHN.T2. PAUL holds SELECT privilege on tables JOHN.T1 and JOHN.T2. MIKE creates view V1 that depends on tables JOHN.T1 and JOHN.T2. The view has an entry for the SELECT privilege in SYSCAT.TABAUTH and two SELECT dependencies in SYSCAT.TABDEP for tables JOHN.T1 and JOHN.T2. Transfer ownership of view V1 to PAUL, who is a regular user.

```
TRANSFER OWNERSHIP OF VIEW V1
  TO USER PAUL PRESERVE PRIVILEGES
```

Ownership of the view is transferred successfully, even though Paul does not hold CONTROL privilege on table JOHN.T2. Paul only needs SELECT privilege on tables JOHN.T1 and JOHN.T2 to maintain the view's existence. (The view only has SELECT privilege because Paul did not hold CONTROL privilege on both tables when the view was created and, as a result, he was not granted CONTROL on the view.) The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'PAUL'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'PAUL'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'PAUL', and TABNAME = 'V1'.

Example 5: Assume that JOHN creates table T1, and that PUBLIC holds SELECT privilege on JOHN.T1. PAUL holds SELECT privilege on JOHN.T1 explicitly, and creates view V1 that depends on table JOHN.T1. Transfer ownership of view V1 to MIKE, who is not a DBADM, but who holds the required privileges to acquire view ownership through a group.

```
TRANSFER OWNERSHIP OF VIEW V1
  TO USER MIKE PRESERVE PRIVILEGES
```

Ownership of the view is transferred successfully, because Mike holds SELECT privilege on table JOHN.T1 through PUBLIC. The value in the OWNER column for the view V1 in the SYSCAT.VIEWS catalog view is replaced with 'MIKE'. The value in the OWNER column for the view V1 in the SYSCAT.TABDEP catalog view is replaced with 'MIKE'. A new row is added to SYSCAT.TABAUTH with the following values: GRANTOR = 'SYSIBM', GRANTEE = 'MIKE', and TABNAME = 'V1'.

Related concepts:

- "Database authorities" on page 74

Part 12. Functions

Chapter 42. Functions

Functions overview

A *function* is an operation that is denoted by a function name followed by a pair of parentheses enclosing the specification of arguments (there may be no arguments).

Built-in functions are provided with the database manager; they return a single result value, and are identified as part of the SYSIBM schema. Built-in functions include column functions (such as AVG), operator functions (such as "+"), casting functions (such as DECIMAL), and others (such as SUBSTR).

User-defined functions are registered to a database in SYSCAT.ROUTINES (using the CREATE FUNCTION statement). User-defined functions are never part of the SYSIBM schema. One such set of functions is provided with the database manager in a schema called SYSFUN, and another in a schema called SYSPROC.

Functions are classified as aggregate (column) functions, scalar functions, row functions, or table functions.

- The argument of an *column function* is a collection of like values. A column function returns a single value (possibly null), and can be specified in an SQL statement wherever an expression can be used.
- The arguments of a *scalar function* are individual scalar values, which can be of different types and have different meanings. A scalar function returns a single value (possibly null), and can be specified in an SQL statement wherever an expression can be used.
- The argument of a *row function* is a structured type. A row function returns a row of built-in data types and can only be specified as a transform function for a structured type.
- The arguments of a *table function* are individual scalar values, which can be of different types and have different meanings. A table function returns a table to the SQL statement, and can be specified only within the FROM clause of a SELECT statement.

The function name, combined with the schema, gives the fully qualified name of a function. The combination of schema, function name, and input parameters make up a *function signature*.

In some cases, the input parameter type is specified as a specific built-in data type, and in other cases, it is specified through a general variable like *any-numeric-type*. If a particular data type is specified, an exact match will only occur with the specified data type. If a general variable is used, each of the data types associated with that variable results in an exact match.

Additional functions may be available, because user-defined functions can be created in different schemas, using one of the function signatures as a source. You can also create external functions in your applications.

Related concepts:

- "Aggregate functions" in *SQL Reference, Volume 1*

Related reference:
- "Functions" in *SQL Reference, Volume 1*
- "Subselect" on page 1211
- "CREATE FUNCTION " on page 920

DBPARTITIONNUM

```
►►──DBPARTITIONNUM──(──column-name──)──────────────────────────────────►◄
```

The schema is SYSIBM.

The DBPARTITIONNUM function returns the database partition number for a row. For example, if used in a SELECT clause, it returns the database partition number for each row in the result set.

The argument must be the qualified or unqualified name of any column in the table. Because row-level information is returned, the result is the same regardless of which column is specified. The column can have any data type.

If *column-name* references a column in a view, the expression for the column in the view must reference a column of the underlying base table, and the view must be deletable. A nested or common table expression follows the same rules as a view.

The specific row (and table) for which the database partition number is returned by the DBPARTITIONNUM function is determined from the context of the SQL statement that uses the function.

The database partition number returned on transition variables and tables is derived from the current transition values of the distribution key columns. For example, in a before insert trigger, the function returns the projected database partition number, given the current values of the new transition variables. However, the values of the distribution key columns might be modified by a subsequent before insert trigger. Thus, the final database partition number of the row when it is inserted into the database might differ from the projected value.

The data type of the result is INTEGER and is never null. If there is no `db2nodes.cfg` file, the result is 0.

This function cannot be used as a source function when creating a user-defined function. Because the function accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.

The DBPARTITIONNUM function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

For compatibility with previous versions of DB2, NODENUMBER can be specified in place of DBPARTITIONNUM.

Examples:
- Count the number of instances in which the row for a given employee in the EMPLOYEE table is on a different database partition than the description of the employee's department in the DEPARTMENT table.

```
SELECT COUNT(*) FROM DEPARTMENT D, EMPLOYEE E
  WHERE D.DEPTNO=E.WORKDEPT
  AND DBPARTITIONNUM(E.LASTNAME) <> DBPARTITIONNUM(D.DEPTNO)
```

- Join the EMPLOYEE and DEPARTMENT tables so that the rows of the two tables are on the same database partition.

```
SELECT * FROM DEPARTMENT D, EMPLOYEE E
  WHERE DBPARTITIONNUM(E.LASTNAME) = DBPARTITIONNUM(D.DEPTNO)
```

- Using a before trigger on the EMPLOYEE table, log the employee number and the projected database partition number of any new row in the EMPLOYEE table in a table named EMPINSERTLOG1.

```
CREATE TRIGGER EMPINSLOGTRIG1
BEFORE INSERT ON EMPLOYEE
REFERENCING NEW AW NEWTABLE
FOR EACH ROW
INSERT INTO EMPINSERTLOG1
VALUES(NEWTABLE.EMPNO, DBPARTITIONNUM
(NEWTABLE.EMPNO))
```

Related reference:

- "CREATE VIEW " on page 1034

DECRYPT_BIN and DECRYPT_CHAR

```
►►──┬─DECRYPT_BIN──┬──(──encrypted-data──┬──────────────────────────────┬──)──►◄
    └─DECRYPT_CHAR─┘                     └─,──password-string-expression─┘
```

The schema is SYSIBM.

The DECRYPT_BIN and DECRYPT_CHAR functions both return a value that is the result of decrypting *encrypted-data*. The password used for decryption is either the *password-string-expression* value or the encryption password value that was assigned by the SET ENCRYPTION PASSWORD statement. The DECRYPT_BIN and DECRYPT_CHAR functions can only decrypt values that are encrypted using the ENCRYPT function (SQLSTATE 428FE).

encrypted-data
: An expression that returns a CHAR FOR BIT DATA or VARCHAR FOR BIT DATA value as a complete, encrypted data string. The data string must have been encrypted using the ENCRYPT function.

password-string-expression
: An expression that returns a CHAR or VARCHAR value with at least 6 bytes and no more than 127 bytes (SQLSTATE 428FC). This expression must be the same password used to encrypt the data (SQLSTATE 428FD). If the value of the password argument is null or not provided, the data will be decrypted using the encryption password value that was assigned for the session by the SET ENCRYPTION PASSWORD statement (SQLSTATE 51039).

The result of the DECRYPT_BIN function is VARCHAR FOR BIT DATA. The result of the DECRYPT_CHAR function is VARCHAR. If *encrypted-data* included a hint, the hint is not returned by the function. The length attribute of the result is the length of the data type of *encrypted-data* minus 8 bytes. The actual length of the value returned by the function will match the length of the original string that was encrypted. If *encrypted-data* includes bytes beyond the encrypted string, these bytes are not returned by the function.

If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

If the data is decrypted on a different system, which uses a code page that is different from the code page in which the data was encrypted, expansion might occur when converting the decrypted value to the database code page. In such situations, the *encrypted-data* value should be cast to a VARCHAR string with a larger number of bytes.

Examples:

- Use the SET ENCRYPTION PASSWORD statement to set an encryption password for the session.

  ```
  CREATE TABLE EMP (SSN VARCHAR(24) FOR BIT DATA);
  SET ENCRYPTION PASSWORD = 'Ben123';
  INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832');
  SELECT DECRYPT_CHAR(SSN)
    FROM EMP;
  ```

  This query returns the value ’289-46-8832’.
- Pass the encryption password explicitly.

  ```
  INSERT INTO EMP (SSN) VALUES ENCRYPT('289-46-8832','Ben123','');
  SELECT DECRYPT_CHAR(SSN,'Ben123')
    FROM EMP;
  ```

  This query returns the value ’289-46-8832’.

Related reference:

- “ENCRYPT ” on page 1266
- “GETHINT ” on page 1268
- “SET ENCRYPTION PASSWORD statement” in *SQL Reference, Volume 2*

ENCRYPT

```
►►──ENCRYPT──────────────────────────────────────────────────────────────────────►

►─(──data-string-expression──┬──────────────────────────────────────────────────┬──)──────────►◄
                             └─,──password-string-expression──┬──────────────────────────┬─┘
                                                              └─,──hint-string-expression─┘
```

The schema is SYSIBM.

The ENCRYPT function returns a value that is the result of encrypting *data-string-expression*. The password used for encryption is either the *password-string-expression* value or the encryption password value that was assigned by the SET ENCRYPTION PASSWORD statement. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

data-string-expression
: An expression that returns a CHAR or a VARCHAR value that is to be encrypted. The length attribute for the data type of *data-string-expression* is limited to 32663 without a *hint-string-expression* argument, and 32631 when the *hint-string-expression* argument is specified (SQLSTATE 42815).

password-string-expression
: An expression that returns a CHAR or a VARCHAR value with at least 6 bytes

and no more than 127 bytes (SQLSTATE 428FC). The value represents the password used to encrypt *data-string-expression*. If the value of the password argument is null or not provided, the data will be encrypted using the encryption password value that was assigned for the session by the SET ENCRYPTION PASSWORD statement (SQLSTATE 51039).

hint-string-expression
: An expression that returns a CHAR or a VARCHAR value with at most 32 bytes that will help data owners remember passwords (for example, 'Ocean' as a hint to remember 'Pacific'). If a hint value is given, the hint is embedded into the result and can be retrieved using the GETHINT function. If this argument is null or not provided, no hint will be embedded in the result.

The result data type of the function is VARCHAR FOR BIT DATA.

- When the optional hint parameter is specified, the length attribute of the result is equal to the length attribute of the unencrypted data + 8 bytes + the number of bytes until the next 8-byte boundary + 32 bytes for the length of the hint.
- When the optional hint parameter is not specified, the length attribute of the result is equal to the length attribute of the unencrypted data + 8 bytes + the number of bytes until the next 8-byte boundary.

If the first argument can be null, the result can be null. If the first argument is null, the result is the null value.

Note that the encrypted result is longer than the *data-string-expression* value. Therefore, when assigning encrypted values, ensure that the target is declared with sufficient size to contain the entire encrypted value.

Notes:

- ***Encryption Algorithm:*** The internal encryption algorithm is RC2 block cipher with padding; the 128-bit secret key is derived from the password using an MD5 message digest.
- ***Encryption Passwords and Data:*** Password management is the user's responsibility. Once the data is encrypted, only the password that was used when encrypting it can be used to decrypt it (SQLSTATE 428FD).

  The encrypted result might contain null terminator and other unprintable characters. Any assignment or cast to a length that is shorter than the suggested data length might result in failed decryption in the future, and lost data. Blanks are valid encrypted data values that might be truncated when stored in a column that is too short.
- ***Administration of encrypted data:*** Encrypted data can only be decrypted on servers that support the decryption functions corresponding to the ENCRYPT function. Therefore, replication of columns with encrypted data should only be done to servers that support the DECRYPT_BIN or the DECRYPT_CHAR function.

Examples:

- Use the SET ENCRYPTION PASSWORD statement to set an encryption password for the session.

  ```
  CREATE TABLE EMP (SSN VARCHAR(24) FOR BIT DATA);
  SET ENCRYPTION PASSWORD = 'Ben123';
  INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832');
  ```
- Pass the encryption password explicitly.

  ```
  INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832','Ben123');
  ```

- Define a password hint.

```
INSERT INTO EMP(SSN) VALUES ENCRYPT('289-46-8832','Pacific','Ocean');
```

Related reference:

- "DECRYPT_BIN and DECRYPT_CHAR " on page 1265
- "GETHINT " on page 1268
- "SET ENCRYPTION PASSWORD statement" in *SQL Reference, Volume 2*

GETHINT

```
►►──GETHINT──(──encrypted-data──)──────────────────────────────────►◄
```

The schema is SYSIBM.

The GETHINT function will return the password hint if one is found in the *encrypted-data*. A password hint is a phrase that will help data owners remember passwords; for example, 'Ocean' as a hint to remember 'Pacific'. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

encrypted-data
: An expression that returns a CHAR FOR BIT DATA or VARCHAR FOR BIT DATA value that is a complete, encrypted data string. The data string must have been encrypted using the ENCRYPT function (SQLSTATE 428FE).

The result of the function is VARCHAR(32). The result can be null; if the hint parameter was not added to the *encrypted-data* by the ENCRYPT function or the first argument is null, the result is the null value.

Example:

In this example the hint 'Ocean' is stored to help the user remember the encryption password 'Pacific'.

```
INSERT INTO EMP (SSN) VALUES ENCRYPT('289-46-8832', 'Pacific','Ocean');
SELECT GETHINT(SSN)
  FROM EMP;
```

The value returned is 'Ocean'.

Related reference:

- "DECRYPT_BIN and DECRYPT_CHAR " on page 1265
- "ENCRYPT " on page 1266

HASHEDVALUE

```
►►──HASHEDVALUE──(──column-name──)─────────────────────────────────►◄
```

The schema is SYSIBM.

The HASHEDVALUE function returns the distribution map index of the row obtained by applying the partitioning function on the distribution key value of the row. For example, if used in a SELECT clause, it returns the distribution map index for each row of the table that was used to form the result of the SELECT statement.

The distribution map index returned on transition variables and tables is derived from the current transition values of the distribution key columns. For example, in a before insert trigger, the function will return the projected distribution map index given the current values of the new transition variables. However, the values of the distribution key columns may be modified by a subsequent before insert trigger. Thus, the final distribution map index of the row when it is inserted into the database may differ from the projected value.

The argument must be the qualified or unqualified name of a column in a table. The column can have any data type. (This function cannot be used as a source function when creating a user-defined function. Because it accepts any data type as an argument, it is not necessary to create additional signatures to support user-defined distinct types.) If *column-name* references a column of a view the expression in the view for the column must reference a column of the underlying base table and the view must be deletable. A nested or common table expression follows the same rules as a view.

The specific row (and table) for which the distribution map index is returned by the HASHEDVALUE function is determined from the context of the SQL statement that uses the function.

The data type of the result is INTEGER in the range 0 to 4095. For a table with no distribution key, the result is always 0. A null value is never returned. Since row-level information is returned, the results are the same, regardless of which column is specified for the table.

The HASHEDVALUE function cannot be used on replicated tables, within check constraints, or in the definition of generated columns (SQLSTATE 42881).

For compatibility with versions earlier than Version 8, the function name PARTITION can be substituted for HASHEDVALUE.

Example:

- List the employee numbers (EMPNO) from the EMPLOYEE table for all rows with a distribution map index of 100.

  ```
  SELECT EMPNO FROM EMPLOYEE
    WHERE HASHEDVALUE(PHONENO) = 100
  ```

- Log the employee number and the projected distribution map index of the new row into a table called EMPINSERTLOG2 for any insertion of employees by creating a before trigger on the table EMPLOYEE.

  ```
  CREATE TRIGGER EMPINSLOGTRIG2
    BEFORE INSERT ON EMPLOYEE
    REFERENCING NEW AW NEWTABLE
    FOR EACH ROW
    INSERT INTO EMPINSERTLOG2
      VALUES(NEWTABLE.EMPNO, HASHEDVALUE(NEWTABLE.EMPNO))
  ```

Related reference:

- “CREATE VIEW ” on page 1034

SECLABEL

```
►►—SECLABEL—(—security-policy-name—,—string-constant—)——————————————►◄
```

The schema is SYSIBM.

The SECLABEL function returns an unnamed security label with a data type of DB2SECURITYLABEL. Use the SECLABEL function to insert a security label with given component values without having to create a named security label.

security-policy-name
: An expression that returns a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC value. This expression must be the name of a security policy that exists at the current server (SQLSTATE 42704).

string-constant
: A character constant that contains a valid representation of a security label for the security policy named by *security-policy-name* (SQLSTATE 4274I).

Examples:

- The following statement inserts a row in table REGIONS which is protected by the security policy named CONTRIBUTIONS. The security label for the row to be inserted is given by the **SECLABEL** function. The security policy CONTRIBUTIONS has two components. The security label given has the element LIFE MEMBER for first component, the elements BLUE and YELLOW for the second component.

```
INSERT INTO REGIONS
VALUES (SECLABEL('CONTRIBUTIONS', 'LIFE MEMBER:(BLUE,YELLOW)') , 1, 'Northeast')
```

- The following statement inserts a row in table CASE_IDS which is protected by the security policy named TS_SECPOLICY, which has three components. The security label is provided by the SECLABEL function. The security label inserted has the element HIGH PROFILE for the first component, the empty value for the second component and the element G19 for the third component.

```
INSERT INTO CASE_IDS
VALUES (SECLABEL('TS_SECPOLICY', 'HIGH PROFILE:():G19') , 3, 'KLB')
```

Related concepts:

- “Built-in functions for dealing with LBAC security labels” on page 127
- “LBAC security labels” on page 118
- “LBAC security policies” on page 111

Related reference:

- “Format for security label values” on page 119
- “SECLABEL_BY_NAME ” on page 1270
- “SECLABEL_TO_CHAR ” on page 1271

SECLABEL_BY_NAME

```
►►—SECLABEL_BY_NAME—(—security-policy-name—,—security-label-name—)——————►◄
```

The schema is SYSIBM.

The SECLABEL_BY_NAME function returns the specified security label. The security label returned has a data type of DB2SECURITYLABEL. Use this function to insert a named security label.

security-policy-name
: A string expression with a resulting type of CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC and a value that is the name of a security policy that exists at the current server (SQLSTATE 42704).

security-label-name
: A string expression with a resulting type of CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC and a value that is the name of a security label for the security policy named by *security-policy-name* (SQLSTATE 4274I).

Examples:

- User Tina is trying to insert a row in table REGIONS which is protected by the security policy named CONTRIBUTIONS. Tina wants the row to be protected by the security label named EMPLOYEESECLABEL. This statement fails because CONTRIBUTIONS.EMPLOYEESECLABEL is an unknown identifier:

  ```
  INSERT INTO REGIONS
  VALUES (CONTRIBUTIONS.EMPLOYEESECLABEL, 1, 'Southwest')   -- incorrect
  ```

  This statement fails because the first value is a string, it does not have a data type of DB2SECURITYLABEL:

  ```
  INSERT INTO REGIONS
  VALUES ('CONTRIBUTIONS.EMPLOYEESECLABEL', 1, 'Southwest') -- incorrect
  ```

  This statement succeeds because the SECLABEL_BY_NAME function returns a security label that has a data type of DB2SECURITYLABEL:

  ```
  INSERT INTO REGIONS
  VALUES (SECLABEL_BY_NAME('CONTRIBUTIONS', 'EMPLOYEESECLABEL'),
    1, 'Southwest')                                         -- correct
  ```

Related concepts:

- "Built-in functions for dealing with LBAC security labels" on page 127
- "LBAC security labels" on page 118
- "LBAC security policies" on page 111

Related reference:

- "SECLABEL " on page 1270
- "SECLABEL_TO_CHAR " on page 1271
- "Format for security label values" on page 119

SECLABEL_TO_CHAR

```
►►──SECLABEL_TO_CHAR──(──security-policy-name──,──security-label──)──────────►◄
```

The schema is SYSIBM.

The SECLABEL_TO_CHAR function accepts a security label and returns a string that contains all elements in the security label. The string is in the security label string format.

security-policy-name
: An expression that returns a CHAR, VARCHAR, GRAPHIC, or VARGRAPHIC value. This expression must be the name of a security policy that exists at the current server (SQLSTATE 42704).

security-label
: An expression that returns a security label value that is valid for the security policy named by *security-policy-name* (SQLSTATE 4274I).

Notes:

- If the authorization ID of the statement executes this function on a security label being read from a column with a data type of DB2SECURITYLABEL then that authorization ID's LBAC credentials might affect the output of the function. In such a case an element is not included in the output if the authorization ID does not have read access to that element. An authorization ID has read access to an element if its LBAC credentials would allow it to read data that was protected by a security label containing only that element, and no others.

  For the rule set DB2LBACRULES only components of type TREE can contain elements that you do not have read access to. For other types of component, if any one of the elements block read access then you will not be able to read the row at all. So only components of type tree will have elements excluded in this way.

Example:

- The EMP table has two columns, RECORDNUM and LABEL; RECORDNUM has data type INTEGER, and LABEL has type DB2SECURITYLABEL. Table EMP is protected by security policy DATA_ACCESSPOLICY, which uses the DB2LBACRULES rule set and has only one component (GROUPS, of type TREE). GROUPS has five elements: PROJECT, TEST, DEVELOPMENT, CURRENT, AND FIELD. The following diagram shows the relationship of these elements to one another:

```
        PROJECT
   ________|________
  |                |
TEST          DEVELOPMENT
               ______|______
              |             |
           CURRENT        FIELD
```

  The EMP table contains the following data:

```
RECORDNUM  LABEL
---------  ----------------
        1  PROJECT
        2  (TEST, FIELD)
        3  (CURRENT, FIELD)
```

  Djavan holds a security label for reading that contains only the DEVELOPMENT element. This means that Djavan has read access to the DEVELOPMENT, CURRENT, and FIELD elements:

```
   SELECT RECORDNUM, SECLABEL_TO_CHAR('DATA_ACCESSPOLICY', LABEL) FROM EMP
```

  returns:

```
RECORDNUM  LABEL
---------  ----------------
        2  FIELD
        3  (CURRENT, FIELD)
```

  The row with a RECORDNUM value of 1 is not included in the output, because Djavan's LBAC credentials do not allow him to read that row. In the row with a

RECORDNUM value of 2, element TEST is not included in the output, because Djavan does not have read access to that element; Djavan would not have been able to access the row at all if TEST were the only element in the security label. Because Djavan has read access to elements CURRENT and FIELD, both elements appear in the output.

Now Djavan is granted an exemption to the DB2LBACREADTREE rule. This means that no element of a TREE type component will block read access. The same query returns:

```
RECORDNUM  LABEL
---------  ----------------
        1  PROJECT
        2  (TEST, FIELD)
        3  (CURRENT, FIELD)
```

This time the output includes all rows and all elements, because the exemption gives Djavan read access to all of the elements.

Related concepts:

- “Built-in functions for dealing with LBAC security labels” on page 127
- “LBAC security labels” on page 118
- “LBAC security policies” on page 111

Related reference:

- “SECLABEL ” on page 1270
- “SECLABEL_BY_NAME ” on page 1270
- “Format for security label values” on page 119

TABLE_NAME

```
►►──TABLE_NAME──(──objectname──┬──────────────────┬──)──────────────────────────►◄
                               └─,──objectschema──┘
```

The schema is SYSIBM.

The TABLE_NAME function returns an unqualified name of the object found after any alias chains have been resolved. The specified *objectname* (and *objectschema*) are used as the starting point of the resolution. If the starting point does not refer to an alias, the unqualified name of the starting point is returned. The resulting name may be of a table, view, or undefined object. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

objectname
: A character expression representing the unqualified name (usually of an existing alias) to be resolved. *objectname* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

objectschema
: A character expression representing the schema used to qualify the supplied *objectname* value before resolution. *objectschema* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

  If *objectschema* is not supplied, the default schema is used for the qualifier.

The data type of the result of the function is VARCHAR(128). If *objectname* can be null, the result can be null; if *objectname* is null, the result is the null value. If *objectschema* is the null value, the default schema name is used. The result is the character string representing an unqualified name. The result name could represent one of the following:

table The value for *objectname* was either a table name (the input value is returned) or an alias name that resolved to the table whose name is returned.

view The value for *objectname* was either a view name (the input value is returned) or an alias name that resolved to the view whose name is returned.

undefined object

The value for *objectname* was either an undefined object (the input value is returned) or an alias name that resolved to the undefined object whose name is returned.

Therefore, if a non-null value is given to this function, a value is always returned, even if no object with the result name exists.

TABLE_SCHEMA

```
►►──TABLE_SCHEMA──(──objectname──┬─────────────────┬──)─────────────────────────►◄
                                 └─,──objectschema─┘
```

The schema is SYSIBM.

The TABLE_SCHEMA function returns the schema name of the object found after any alias chains have been resolved. The specified *objectname* (and *objectschema*) are used as the starting point of the resolution. If the starting point does not refer to an alias, the schema name of the starting point is returned. The resulting schema name may be of a table, view, or undefined object. In a Unicode database, if a supplied argument is a graphic string, it is first converted to a character string before the function is executed.

objectname
A character expression representing the unqualified name (usually of an existing alias) to be resolved. *objectname* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

objectschema
A character expression representing the schema used to qualify the supplied *objectname* value before resolution. *objectschema* must have a data type of CHAR or VARCHAR and a length greater than 0 and less than 129 bytes.

If *objectschema* is not supplied, the default schema is used for the qualifier.

The data type of the result of the function is VARCHAR(128). If *objectname* can be null, the result can be null; if *objectname* is null, the result is the null value. If *objectschema* is the null value, the default schema name is used. The result is the character string representing a schema name. The result schema could represent the schema name for one of the following:

table The value for *objectname* was either a table name (the input or default value of *objectschema* is returned) or an alias name that resolved to a table for which the schema name is returned.

view The value for *objectname* was either a view name (the input or default value of *objectschema* is returned) or an alias name that resolved to a view for which the schema name is returned.

undefined object
The value for *objectname* was either an undefined object (the input or default value of *objectschema* is returned) or an alias name that resolved to an undefined object for which the schema name is returned.

Therefore, if a non-null *objectname* value is given to this function, a value is always returned, even if the object name with the result schema name does not exist. For example, `TABLE_SCHEMA('DEPT', 'PEOPLE')` returns 'PEOPLE ' if the catalog entry is not found.

Examples:

- PBIRD tries to select the statistics for a given table from SYSCAT.TABLES using an alias PBIRD.A1 defined on the table HEDGES.T1.

  ```
  SELECT NPAGES, CARD FROM SYSCAT.TABLES
    WHERE TABNAME = TABLE_NAME ('A1')
    AND TABSCHEMA = TABLE_SCHEMA ('A1')
  ```

  The requested statistics for HEDGES.T1 are retrieved from the catalog.
- Select the statistics for an object called HEDGES.X1 from SYSCAT.TABLES using HEDGES.X1. Use TABLE_NAME and TABLE_SCHEMA since it is not known whether HEDGES.X1 is an alias or a table.

  ```
  SELECT NPAGES, CARD FROM SYSCAT.TABLES
    WHERE TABNAME = TABLE_NAME ('X1','HEDGES')
    AND TABSCHEMA = TABLE_SCHEMA ('X1','HEDGES')
  ```

  Assuming that HEDGES.X1 is a table, the requested statistics for HEDGES.X1 are retrieved from the catalog.
- Select the statistics for a given table from SYSCAT.TABLES using an alias PBIRD.A2 defined on HEDGES.T2 where HEDGES.T2 does not exist.

  ```
  SELECT NPAGES, CARD FROM SYSCAT.TABLES
    WHERE TABNAME = TABLE_NAME ('A2','PBIRD')
    AND TABSCHEMA = TABLE_SCHEMA ('A2',PBIRD')
  ```

  The statement returns 0 records as no matching entry is found in SYSCAT.TABLES where TABNAME = 'T2' and TABSCHEMA = 'HEDGES'.
- Select the qualified name of each entry in SYSCAT.TABLES along with the final referenced name for any alias entry.

  ```
  SELECT TABSCHEMA AS SCHEMA,  TABNAME AS NAME,
    TABLE_SCHEMA (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_SCHEMA,
    TABLE_NAME (BASE_TABNAME, BASE_TABSCHEMA) AS REAL_NAME
    FROM SYSCAT.TABLES
  ```

  The statement returns the qualified name for each object in the catalog and the final referenced name (after alias has been resolved) for any alias entries. For all non-alias entries, BASE_TABNAME and BASE_TABSCHEMA are null so the REAL_SCHEMA and REAL_NAME columns will contain nulls.

Part 13. Applications

Chapter 43. Application considerations

About SQL statements

How SQL statements are invoked

SQL statements are classified as executable or non-executable.

An *executable statement* can be invoked in four ways. It can be:
- Embedded in an application program
- Embedded in an SQL procedure.
- Prepared and executed dynamically
- Issued interactively

Depending on the statement, some or all of these methods can be used. (Statements embedded in REXX are prepared and executed dynamically.)

A *non-executable statement* can only be embedded in an application program.

Another SQL statement construct is the select-statement. A *select-statement* can be invoked in three ways. It can be:
- Included in DECLARE CURSOR, and executed implicitly by OPEN, FETCH and CLOSE (static invocation)
- Prepared dynamically, referenced in DECLARE CURSOR, and executed implicitly by OPEN, FETCH and CLOSE (dynamic invocation)
- Issued interactively

Embedding a statement in an application program

SQL statements can be included in a source program that will be submitted to a precompiler. Such statements are said to be *embedded* in the program. An embedded statement can be placed anywhere in the program where a host language statement is allowed. Each embedded statement must be preceded by the keywords EXEC SQL.

Executable statements: An executable statement embedded in an application program is executed every time a statement of the host language would be executed if it were specified in the same place. Thus, a statement within a loop is executed every time the loop is executed, and a statement within a conditional construct is executed only when the condition is satisfied.

An embedded statement can contain references to host variables. A host variable referenced in this way can be used in two ways. It can be used:
- As input (the current value of the host variable is used in the execution of the statement)
- As output (the variable is assigned a new value as a result of executing the statement)

In particular, all references to host variables in expressions and predicates are effectively replaced by current values of the variables; that is, the variables are used as input.

All executable statements should be followed by a test of the SQL return code. Alternatively, the WHENEVER statement (which is itself non-executable) can be used to change the flow of control immediately after the execution of an embedded statement.

All objects referenced in data manipulation language (DML) statements must exist when the statements are bound to a database.

Non-executable statements: An embedded non-executable statement is processed only by the precompiler. The precompiler reports any errors encountered in the statement. The statement is *never* processed during program execution; therefore, such statements should not be followed by a test of the SQL return code.

Embedding a statement in an SQL procedure: Statements can be included in the SQL-procedure-body portion of the CREATE PROCEDURE statement. Such statements are said to be embedded in the SQL procedure. Whenever an SQL statement description refers to a *host-variable*, an *SQL-variable* can be used if the statement is embedded in an SQL procedure.

Dynamic preparation and execution

An application program can dynamically build an SQL statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the program (for example, input from a workstation). The statement (not a select-statement) constructed can be prepared for execution by means of the (embedded) PREPARE statement, and executed by means of the (embedded) EXECUTE statement. Alternatively, an (embedded) EXECUTE IMMEDIATE statement can be used to prepare and execute the statement in one step.

A statement that is going to be dynamically prepared must not contain references to host variables. It can instead contain parameter markers. (For rules concerning parameter markers, see “PREPARE”.) When the prepared statement is executed, the parameter markers are effectively replaced by current values of the host variables specified in the EXECUTE statement. Once prepared, a statement can be executed several times with different values for the host variables. Parameter markers are not allowed in the EXECUTE IMMEDIATE statement.

Successful or unsuccessful execution of the statement is indicated by the setting of an SQL return code in the SQLCA after the EXECUTE (or EXECUTE IMMEDIATE) statement completes. The SQL return code should be checked, as described above. For more information, see “SQL return codes” on page 1282.

Static invocation of a select-statement

A select-statement can be included as a part of the (non-executable) DECLARE CURSOR statement. Such a statement is executed every time the cursor is opened by means of the (embedded) OPEN statement. After the cursor is open, the result table can be retrieved, one row at a time, by successive executions of the FETCH statement.

Used in this way, the select-statement can contain references to host variables. These references are effectively replaced by the values that the variables have when the OPEN statement executes.

Dynamic invocation of a select-statement

An application program can dynamically build a select-statement in the form of a character string placed in a host variable. In general, the statement is built from some data available to the program (for example, a query obtained from a workstation). The statement so constructed can be prepared for execution by means of the (embedded) PREPARE statement, and referenced by a (non-executable) DECLARE CURSOR statement. The statement is then executed every time the cursor is opened by means of the (embedded) OPEN statement. After the cursor is open, the result table can be retrieved, one row at a time, by successive executions of the FETCH statement.

Used in this way, the select-statement must not contain references to host variables. It can contain parameter markers instead. The parameter markers are effectively replaced by the values of the host variables specified in the OPEN statement.

Interactive invocation

A capability for entering SQL statements from a workstation is part of the architecture of the database manager. A statement entered in this way is said to be issued interactively. Such a statement must be an executable statement that does not contain parameter markers or references to host variables, because these make sense only in the context of an application program.

SQL use with other host systems

SQL statement syntax exhibits minor variations among different types of host systems (DB2 for z/OS, DB2 for iSeries, DB2 Database for Linux, UNIX, and Windows). Regardless of whether the SQL statements in an application are static or dynamic, it is important — if the application is meant to access different database host systems — to ensure that the SQL statements and precompile/bind options are supported on the database systems that the application will access.

Further information about SQL statements used in other host systems can be found in the *DB2 Universal Database for iSeries SQL Reference* and the *DB2 Universal Database for OS/390 and z/OS SQL Reference*.

SQL return codes

An application program containing executable SQL statements can use either SQLCODE or SQLSTATE values to handle return codes from SQL statements. There are two ways in which an application can get access to these values.

- Include a structure named SQLCA. The SQLCA includes an integer variable named SQLCODE and a character string variable named SQLSTATE. In REXX, an SQLCA is provided automatically. In other languages, an SQLCA can be obtained by using the INCLUDE SQLCA statement.
- If LANGLEVEL SQL92E is specified as a precompile option, a variable named SQLCODE or SQLSTATE can be declared in the SQL declare section of the program. If neither of these variables is declared in the SQL declare section, it is

assumed that a variable named SQLCODE is declared elsewhere in the program. With LANGLEVEL SQL92E, the program should not have an INCLUDE SQLCA statement.

SQLCODE: An SQLCODE is set by the database manager after each SQL statement executes. All database managers conform to the ISO/ANSI SQL standard, as follows:

- If SQLCODE = 0 and SQLWARN0 is blank, execution was successful.
- If SQLCODE = 100, “no data” was found. For example, a FETCH statement returned no data, because the cursor was positioned after the last row of the result table.
- If SQLCODE > 0 and not = 100, execution was successful with a warning.
- If SQLCODE = 0 and SQLWARN0 = 'W', execution was successful, but one or more warning indicators were set.
- If SQLCODE < 0, execution was not successful.

The meaning of SQLCODE values other than 0 and 100 is product-specific.

SQLSTATE: An SQLSTATE is set by the database manager after each SQL statement executes. Application programs can check the execution of SQL statements by testing SQLSTATE instead of SQLCODE. SQLSTATE provides common codes for common error conditions. Application programs can test for specific errors or classes of errors. The coding scheme is the same for all IBM database managers, and is based on the ISO/ANSI SQL92 standard.

SQL comments

Static SQL statements can include host language or SQL comments. Dynamic SQL statements can include SQL comments. There are two types of SQL comments:

simple comments
: Simple comments are introduced by two consecutive hyphens (--) and end with the end of line.

bracketed comments
: Bracketed comments are introduced by /* and end with */.

The following rules apply to the use of simple comments:

- The two hyphens must be on the same line and must not be separated by a space.
- Simple comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').
- Simple comments cannot be continued to the next line.
- In COBOL, the hyphens must be preceded by a space.

The following rules apply to the use of bracketed comments:

- The /* must be on the same line and must not be separated by a space.
- The */ must be on the same line and must not be separated by a space.
- Bracketed comments can be started wherever a space is valid (except within a delimiter token or between 'EXEC' and 'SQL').
- Bracketed comments can be continued to subsequent lines.

Example 1: This example shows how to include simple comments in a statement:

```
CREATE VIEW PRJ_MAXPER       -- PROJECTS WITH MOST SUPPORT PERSONNEL
  AS SELECT PROJNO, PROJNAME -- NUMBER AND NAME OF PROJECT
    FROM PROJECT
    WHERE DEPTNO = 'E21'     -- SYSTEMS SUPPORT DEPT CODE
    AND PRSTAFF > 1
```

Example 2: This example shows how to include bracketed comments in a statement:

```
CREATE VIEW PRJ_MAXPER       /* PROJECTS WITH MOST SUPPORT
                                  PERSONNEL                */
  AS SELECT PROJNO, PROJNAME /* NUMBER AND NAME OF PROJECT  */
    FROM PROJECT
    WHERE DEPTNO = 'E21'     /* SYSTEMS SUPPORT DEPT CODE   */
    AND PRSTAFF > 1
```

Related reference:

- "Select-statement" on page 1182
- "SQLCA (SQL communications area)" on page 1357
- "EXECUTE statement" in *SQL Reference, Volume 2*
- "OPEN statement" in *SQL Reference, Volume 2*
- "PREPARE statement" in *SQL Reference, Volume 2*

About SQL control statements

Control statements are SQL statements that allow structured query language to be used in a manner similar to writing a program in a structured programming language. SQL control statements provide the capability to control the logic flow, declare, and set variables, and handle warnings and exceptions. Some SQL control statements include other nested SQL statements. SQL control statements can be used in the body of a routine, trigger or a dynamic compound statement.

References to SQL parameters and SQL variables:

SQL parameters and SQL variables can be referenced anywhere in an SQL procedure statement where an expression or variable can be specified. Host variables cannot be specified in SQL routines, SQL triggers or dynamic compound statements. SQL parameters can be referenced anywhere in the routine, and can be qualified with the routine name. SQL variables can be referenced anywhere in the compound statement in which they are declared, and can be qualified with the label name specified at the beginning of the compound statement.

All SQL parameters and SQL variables are considered nullable. The name of an SQL parameter or SQL variable in an SQL routine can be the same as the name of a column in a table or view referenced in the routine. The name of an SQL variable can also be the same as the name of another SQL variable declared in the same routine. This can occur when the two SQL variables are declared in different compound statements. The compound statement that contains the declaration of an SQL variable determines the scope of that variable. For more information, see "Compound SQL (Procedure)".

Names that are the same should be explicitly qualified. Qualifying a name clearly indicates whether the name refers to a column, SQL variable, or SQL parameter. If the name is not qualified, or qualified but still ambiguous, the following rules describe whether the name refers to the column or to the SQL variable or SQL parameter:

- If the tables and views specified in an SQL routine body exist at the time the routine is created, the name is first checked as a column name. If not found as a column, it is then checked as an SQL variable in the compound statement, and then checked as an SQL parameter.
- If the referenced tables or views do not exist at the time the routine is created, the name is first checked as an SQL variable in the compound statement and then as an SQL parameter. The variable can be declared within the compound statement that contains the reference, or within a compound statement in which that compound statement is nested. If two SQL variables are within the same scope and have the same name, which can happen if they are declared in different compound statements, the SQL variable that is declared in the innermost compound statement is used. If not found, it is assumed to be a column.

The name of an SQL variable or SQL parameter in an SQL routine can be the same as the name of an identifier used in certain SQL statements. If the name is not qualified, the following rules describe whether the name refers to the identifier or to the SQL parameter or SQL variable:

- In the SET PATH and SET SCHEMA statements, the name is checked as an SQL parameter or SQL variable. If not found as an SQL variable or SQL parameter, it is used as an identifier.
- In the CONNECT, DISCONNECT, RELEASE, and SET CONNECTION statements, the name is used as an identifier.

References to labels:

Labels can be specified on most SQL procedure statements. The compound statement that contains the statement that defines a label determines the scope of that label name. A label name must be unique within the compound statement in which it is defined, including any labels defined in compound statements that are nested within that compound statement (SQLSTATE 42734). The label must not be the same as a label specified on the compound statement itself (SQLSTATE 42734), or the same as the name of the routine that contains the SQL procedure statement (SQLSTATE 42734).

A label name can only be referenced within the compound statement in which it is defined, including any compound statements that are nested within that compound statement. A label can be used to qualify the name of an SQL variable, or it can be specified as the target of a GOTO, LEAVE, or ITERATE statement.

References to SQL condition names:

The name of an SQL condition can be the same as the name of another SQL condition declared in the same routine. This can occur when the two SQL conditions are declared in different compound statements. The compound statement that contains the declaration of an SQL condition name determines the scope of that condition name. A condition name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A condition name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a condition name, the condition that is declared in the innermost compound statement is the condition that is used. For more information, see “Compound SQL (Procedure)”.

References to SQL statement names:

The name of an SQL statement can be the same as the name of another SQL statement declared in the same routine. This can occur when the two SQL statements are declared in different compound statements. The compound statement that contains the declaration of an SQL statement name determines the scope of that statement name. A statement name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A statement name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a statement name, the statement that is declared in the innermost compound statement is the statement that is used. For more information, see "Compound SQL (Procedure)".

References to SQL cursor names:

The name of an SQL cursor can be the same as the name of another SQL cursor declared in the same routine. This can occur when the two SQL cursors are declared in different compound statements. The compound statement that contains the declaration of an SQL cursor determines the scope of that cursor name. A cursor name must be unique within the compound statement in which it is declared, excluding any declarations in compound statements that are nested within that compound statement (SQLSTATE 42734). A cursor name can only be referenced within the compound statement in which it is declared, including any compound statements that are nested within that compound statement. When there is a reference to a cursor name, the cursor that is declared in the innermost compound statement is the cursor that is used. For more information, see "Compound SQL (Procedure)".

Related reference:
- "Compound SQL (Procedure) statement" in *SQL Reference, Volume 2*

Function, method, and procedure designators

The following sections describe syntax fragments that are used to uniquely identify a function, method, or procedure. The fragments are referenced as follows:

```
>>──┤ fragment ├────────────────────────────────────────────────><
```

Function designator

A function designator uniquely identifies a single function. Function designators typically appear in DDL statements for functions (such as DROP or ALTER).

Syntax:

function-designator:

```
├──┬─FUNCTION──function-name──┬────────────────────────────┬─┬──────────┤
   │                          └─(──┬──────────────────┬──)─┘ │
   │                               │    ┌─,─────────┐ │      │
   │                               │    ▼           │ │      │
   │                               └─(────data-type─┴──)─┘   │
   └─SPECIFIC FUNCTION──specific-name────────────────────────┘
```

Description:

FUNCTION *function-name*
: Identifies a particular function, and is valid only if there is exactly one function instance with the name *function-name* in the schema. The identified function can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no function by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one instance of the function in the named or implied schema, an error (SQLSTATE 42725) is raised.

FUNCTION *function-name (data-type,...)*
: Provides the function signature, which uniquely identifies the function. The function resolution algorithm is not used.

function-name
: Specifies the name of the function. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

(data-type,...)
: Values must match the data types that were specified (in the corresponding position) on the CREATE FUNCTION statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific function instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE FUNCTION statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because $0 < n < 25$ means REAL, and $24 < n < 54$ means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no function with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC FUNCTION *specific-name*
: Identifies a particular user-defined function, using the name that is specified or defaulted to at function creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific function instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

Method designator

A method designator uniquely identifies a single method. Method designators typically appear in DDL statements for methods (such as DROP or ALTER).

Syntax:

method-designator:

```
├──┬─METHOD──method-name──┬──────────────────────────────┬─FOR──type-name─┬──────────┤
   │                      └─(─┬──────────────────────┬─)─┘                │
   │                          │     ┌─,─────────┐    │                    │
   │                          └─(───▼─data-type─┴─)──┘                    │
   └─SPECIFIC METHOD──specific-name───────────────────────────────────────┘
```

Description:

METHOD *method-name*
: Identifies a particular method, and is valid only if there is exactly one method instance with the name *method-name* for the type *type-name*. The identified method can have any number of parameters defined for it. If no method by this name exists for the type, an error (SQLSTATE 42704) is raised. If there is more than one instance of the method for the type, an error (SQLSTATE 42725) is raised.

METHOD *method-name (data-type,...)*
: Provides the method signature, which uniquely identifies the method. The method resolution algorithm is not used.

method-name
: Specifies the name of the method for the type *type-name*.

(data-type,...)
: Values must match the data types that were specified (in the corresponding position) on the CREATE TYPE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific method instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE TYPE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because $0 < n < 25$ means REAL, and $24 < n < 54$ means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no method with the specified signature exists for the type in the named or implied schema, an error (SQLSTATE 42883) is raised.

FOR *type-name*
: Names the type with which the specified method is to be associated. The name must identify a type already described in the catalog (SQLSTATE

42704). In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

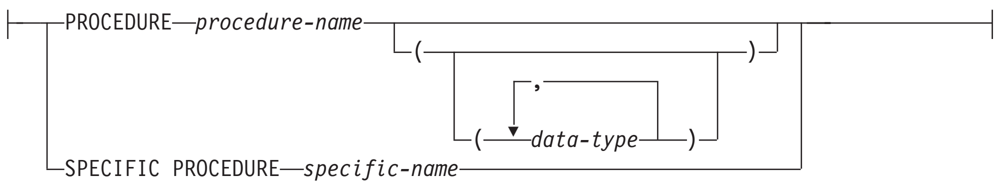
SPECIFIC METHOD *specific-name*
: Identifies a particular method, using the name that is specified or defaulted to at method creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific method instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

Procedure designator

A procedure designator uniquely identifies a single procedure. Procedure designators typically appear in DDL statements for procedures (such as DROP or ALTER).

Syntax:

procedure-designator:



Description:

PROCEDURE *procedure-name*
: Identifies a particular procedure, and is valid only if there is exactly one procedure instance with the name *procedure-name* in the schema. The identified procedure can have any number of parameters defined for it. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. If no procedure by this name exists in the named or implied schema, an error (SQLSTATE 42704) is raised. If there is more than one instance of the procedure in the named or implied schema, an error (SQLSTATE 42725) is raised.

PROCEDURE *procedure-name (data-type,...)*
: Provides the procedure signature, which uniquely identifies the procedure. The procedure resolution algorithm is not used.

  procedure-name
  : Specifies the name of the procedure. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names.

  (data-type,...)
  : Values must match the data types that were specified (in the corresponding

position) on the CREATE PROCEDURE statement. The number of data types, and the logical concatenation of the data types, is used to identify the specific procedure instance.

If a data type is unqualified, the type name is resolved by searching the schemas on the SQL path. This also applies to data type names specified for a REFERENCE type.

It is not necessary to specify the length, precision, or scale for the parameterized data types. Instead, an empty set of parentheses can be coded to indicate that these attributes are to be ignored when looking for a data type match.

FLOAT() cannot be used (SQLSTATE 42601), because the parameter value indicates different data types (REAL or DOUBLE).

If length, precision, or scale is coded, the value must exactly match that specified in the CREATE PROCEDURE statement.

A type of FLOAT(*n*) does not need to match the defined value for *n*, because 0 < *n* < 25 means REAL, and 24 < *n* < 54 means DOUBLE. Matching occurs on the basis of whether the type is REAL or DOUBLE.

If no procedure with the specified signature exists in the named or implied schema, an error (SQLSTATE 42883) is raised.

SPECIFIC PROCEDURE *specific-name*
Identifies a particular procedure, using the name that is specified or defaulted to at procedure creation time. In dynamic SQL statements, the CURRENT SCHEMA special register is used as a qualifier for an unqualified object name. In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified object names. The *specific-name* must identify a specific procedure instance in the named or implied schema; otherwise, an error (SQLSTATE 42704) is raised.

Database connection management via embedded SQL applications

Connecting to DB2 databases in embedded SQL applications

Before working with a database, you need to establish a connection to that database. Embedded SQL provides multiple ways in which to include code for establishing database connections. Depending on the embedded SQL host programming language there might be one or more way of doing this.

Database connections can be established implicitly or explicitly. An implicit connection is a connection where the user ID is presumed to be the current user ID. This type of connection is not recommended for database applications. Explicit database connections, which require that a user ID and password be specified, are strongly recommended.

Connecting to DB2 databases in C and C++ Embedded SQL applications:

When working with C and C++ applications, a database connection can be established by executing the following statement.

```
EXEC SQL CONNECT TO sample;
```

If you want to use a specific user id (`herrick`) and password (`mypassword`), use the following statement:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword;
```

Connecting to DB2 databases in COBOL Embedded SQL applications:

When working with COBOL applications, a database connection is established by executing the following statement. This statement creates a connection to the sample database using the default user name.

```
EXEC SQL CONNECT TO sample END-EXEC.
```

If you want to use a specific user id (`herrick`) and password (`mypassword`), use the following statement:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword END-EXEC.
```

Connecting to DB2 databases in FORTRAN Embedded SQL applications:

When working with FORTRAN applications, a database connection is established by executing the following statement. This statement creates a connection to the sample database using the default user name.

```
EXEC SQL CONNECT TO sample
```

If you want to use a specific user id (`herrick`) and password (`mypassword`), use the following statement:

```
EXEC SQL CONNECT TO sample USER herrick USING mypassword
```

Connecting to DB2 databases in REXX Embedded SQL applications:

When working with REXX applications, a database connection is established by executing the following statement. This statement creates a connection to the sample database using the default user name.

```
CALL SQLEXEC 'CONNECT TO sample'
```

If you want to use a specific user id (`herrick`) and password (`mypassword`), use the following statement:

```
CALL SQLEXEC 'CONNECT TO sample USER herrick USING mypassword'
```

Related concepts:

- "Executing SQL statements in embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:

- "CONNECT (Type 1) " on page 1159
- "CONNECT (Type 2) " on page 1165
- "EXECUTE statement" in *SQL Reference, Volume 2*

Disconnecting from embedded SQL applications

The disconnect statement is the final step in working with a database. This topic will provide examples of the disconnect statement in the supported host languages.

Disconnecting from DB2 databases in C and C++ Embedded SQL applications:

When working with C and C++ applications, a database connection is closed by issuing the following statement:

```
EXEC SQL CONNECT RESET;
```

Disconnecting from DB2 databases in COBOL Embedded SQL applications:

When working with COBOL applications, a database connection is closed by issuing the following statement:

```
EXEC SQL CONNECT RESET END-EXEC.
```

Disconnecting from DB2 databases in REXX Embedded SQL applications:

When working with REXX applications, a database connection is closed by issuing the following statement:

```
CALL SQLEXEC 'CONNECT RESET'
```

When working with FORTRAN applications, a database connection is closed by issuing the following statement:

```
EXEC SQL CONNECT RESET
```

Related concepts:

- "Embedding SQL statements in a host language" in *Developing Embedded SQL Applications*
- "Executing SQL statements in embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:

- "DISCONNECT " on page 1172

Considerations for routines

Security of routines

The security of routines is paramount to ensure their continued functioning, to minimize the risk of tampering, and to protect the database system environment. There are a few categories of routine security considerations each with varying levels of risk. One must be aware of these risks when developing or maintaining routines so as to mitigate unfortunate outcomes as much as possible.

Security control of who can create routines:

The security of routines begins when users are given the necessary privileges to execute the CREATE statement required to create routines in the database. When granting these privileges, it is important to understand the corresponding risks:

- Users with the privilege to execute the CREATE statement for a routine can create multiple routines.
- Users with the privilege to execute the CREATE statement for a routine can create routines that can modify the database layout or database data subject to the other privileges that user has.
- Users that successfully create routines are automatically granted the EXECUTE privilege required to invoke the routine.
- Users that successfully create routines are automatically granted the ALTER ROUTINE privilege required to modify the routine.

To minimize the risk of users modifying the database and data:

- Minimize the number of users that have the privilege to create routines.

- Ensure that the user IDs of departed employees are removed, or if they are re-used, be sure to assess the procedure related privileges.

Refer to the topics on controlling access to database objects and data for more on how to grant and revoke privileges from one, many, or all database users.

Security control of who can invoke routines:

It is easy to determine when users require privileges: they are unable to do something. It is harder to determine when users no longer require these privileges. This is particularly true when it comes to users with privileges to invoke routines, as allowing them to retain their privileges can introduce risks:

- Users that have been granted the EXECUTE privilege to invoke a routine will continue to be able to invoke the routine until this privilege is removed. If the routine contains sensitive logic or acts on sensitive data this can be a business risk.

To minimize the risk of users modifying the database and data:

- Minimize the number of users that have the privilege to invoke routines.
- Ensure that the user IDs of departed employees are removed, or if they are re-used, be sure to assess the procedure related privileges.
- If you suspect that someone is maliciously invoking routines, you should revoke the EXECUTE privilege for each of those routines.

Security control of routines defined with FENCED or NOT FENCED clauses:

When formulating the CREATE statement for a routine, you must determine whether you want to specify the FENCED clause or NOT FENCED clause. Once you understand the benefits of creating a routine as fenced or unfenced it is important to assess the risks associated with running routines with external implementations as NOT FENCED.

- Routines created with the NOT FENCED clause can accidentally or maliciously corrupt the database manager's shared memory, damage the database control structures, or access database manager resources which can cause the database manager to fail. There is also the risk that they will corrupt databases and their tables.

To ensure the integrity of the database manager and its databases:

- Thoroughly screen routines you intend to create that specify the NOT FENCED clause. These routines must be fully tested, debugged, and not exhibit any unexpected side-effects. In the examination of the routine code, pay close attention to memory management and the use of static variables. The greatest potential for corruption arises when code does not properly manage memory or incorrectly uses static variables. These problems are prevalent in languages other than Java(TM) and .NET programming languages.

In order to register a NOT FENCED routine, the CREATE_NOT_FENCED_ROUTINE authority is required. When granting the CREATE_NOT_FENCED_ROUTINE authority, be aware that the recipient can potentially gain unrestricted access to the database manager and all its resources.

Note: NOT FENCED routines are not supported in Common Criteria compliant configurations.

Related concepts:
- "Overview of routines" in *Developing SQL and External Routines*
- "Security and execution modes for CLR routines" in *Developing SQL and External Routines*
- "Security of external routine library or class files" in *Developing SQL and External Routines*

Related tasks:
- "Securing routines" on page 1294

Securing routines

When creating routines it is important to ensure that the routines, routine libraries (in the case of external routines), and the privileges of the users that will interact with the routines are managed with routine security in mind.

Although it might not be necessary to have anything as elaborate as a routine security strategy, it helps to be mindful of the factors contributing to the security of routines and to follow a disciplined approach when securing routines.

Prerequisites:
- Read the topic, "Security of routines".
- To fully secure routines within the database system you must have:
  - Root user access on the database server operating system.
  - One of the DBADM or SYSADM authorities.

Procedure:

Whether you are creating a routine, or assessing an existing routine, the procedure for securing a routine is similar.
1. Limit the number of user IDs with the privileges required to create routines and ensure that these users are allowed to have these privileges.
   - Upon successful execution of the CREATE statement for a routine, this user ID will automatically be granted other privileges including the EXECUTE privilege, which allows the user to invoke the routine, and the GRANT EXECUTE privilege, which allows the user to grant the ability to invoke the routine to other users.
   - Ensure that the users with this privilege are few and that the right users get this privilege.
2. Assess the routine for potentially malicious or inadequately reviewed or tested code.
   - Consider the origin of the routine. Is the party that supplied the routine reliable?
   - Look for malicious code such as code that attempts to read or write to the database server file system and or replace files there.
   - Look for poorly implemented code related to memory management, pointer manipulation, and the use of static variables that might cause the routine to fail.
   - Verify that the code has been adequately tested.
3. Reject routines that appear to be excessively unsafe or poorly coded - the risk is not always worth it.

4. Contain the risks associated with only somewhat potentially risky routines.
   - SQL user-defined SQL routines are by default created as NOT FENCED THREADSAFE routines, because they are safe to run within the database manager memory space. For these routines you do not need to do anything.
   - Specify the FENCED clause in the CREATE statement for the routine. This will ensure that the routine operation does not affect the database manager. This is a default clause.
   - If the routine is multi-threaded, specify the NOT THREADSAFE clause in the CREATE statement for the routine. This will ensure that any failures or malicious code in the routine do not impact other routines that might run in a shared thread process.
5. If the routine is an external routine, you must put the routine implementation library or class file on the database server. Follow the general recommendations for deploying routines and the specific recommendations for deploying external routine library or class files.

Related concepts:
- "Security of routines" on page 1292
- "Security and execution modes for CLR routines" in *Developing SQL and External Routines*
- "Security of external routine library or class files" in *Developing SQL and External Routines*

Guidelines for stored procedures

Stored procedures permit one call to a remote database to execute a preprogrammed procedure in a database application environment in which many situations are repetitive. For example, for receiving a fixed set of data, performing the same set of multiple requests against a database, or returning a fixed set of data might represent several accesses to the database.

Processing a single SQL statement for a remote database requires sending two transmissions: one request and one receive. Because an application contains many SQL statements it requires many transmissions to complete its work.

However, when a database client uses a stored procedure that encapsulates many SQL statements, it requires only two transmissions for the entire process.

Stored procedures usually run in processes separate from the database agents. This separation requires the stored procedure and agent processes to communicate through a router. However, a special kind of stored procedure that runs in the agent process might improve performance, although it carries significant risks of corrupting data and databases.

These risky stored procedures are those created as *not fenced*. For a not-fenced stored procedure, nothing separates the stored procedure from the database control structures that the database agent uses. If a DBA wants to ensure that the stored procedure operations will not accidentally or maliciously damage the database control structures, the *not fenced* option is omitted.

Because of the risk of damaging your database, use *not fenced* stored procedures **only** when you need the maximum possible performance benefits. In addition, make absolutely sure that the procedure is well coded and has been thoroughly tested before allowing it to run as a not-fenced stored procedure. If a fatal error occurs while running a not-fenced stored procedure, the database manager determines whether the error occurred in the application or database manager code and performs the appropriate recovery.

A not-fenced stored procedure can corrupt the database manager beyond recovery, possibly resulting in lost data and the possibility of a corrupt database. Exercise extreme caution when you run not-fenced trusted stored procedures. In almost all cases, the proper performance analysis of an application results in the good performance without using not-fenced stored procedures. For example, triggers might improve performance.

Related concepts:
- "Query tuning guidelines" in *Performance Guide*

Privileges required to query or modify table data

To successfully query and modify data, users must have the necessary privileges or authority to perform these types of operations. There is no single privilege which grants a user access to query and modify all data in a database; however users can be granted special database authority roles which can give them a broader set or a complete set of privileges on the database.

Querying data, in general, requires:
- the SELECT privilege on the table or set of tables containing the data
- the privilege to execute the particular SELECT statement
- the privileges required to reference other database objects within the query

For the specific privileges required to execute queries, refer to the SQL Reference topics on the fullselect and the SELECT statements.

Modifying data, in general requires:
- One of INSERT, UPDATE, or DELETE privilege on the table or set of tables containing the data
- the privilege to execute the particular data change statement (INSERT, UPDATE, DELETE)
- the privileges required to reference other database objects within the query

For the specific privileges required to modify data, refer to the SQL Reference topics on the INSERT, UPDATE, and DELETE statements. Other SQL statements can result in data being modified including the MERGE statement and the CALL statement.

The degree to which a user should be granted access to query and modify data depends on a company's security policy and the job function of the user. It is prudent to limit user access to query and modify data to only the sets of data that they require to perform their job.

Security Considerations when Using SQL in Applications

Package Creation for Embedded SQL

To run applications written in compiled host languages, you must create the packages needed by the database manager at execution time. This involves the following steps as shown in the following figure:

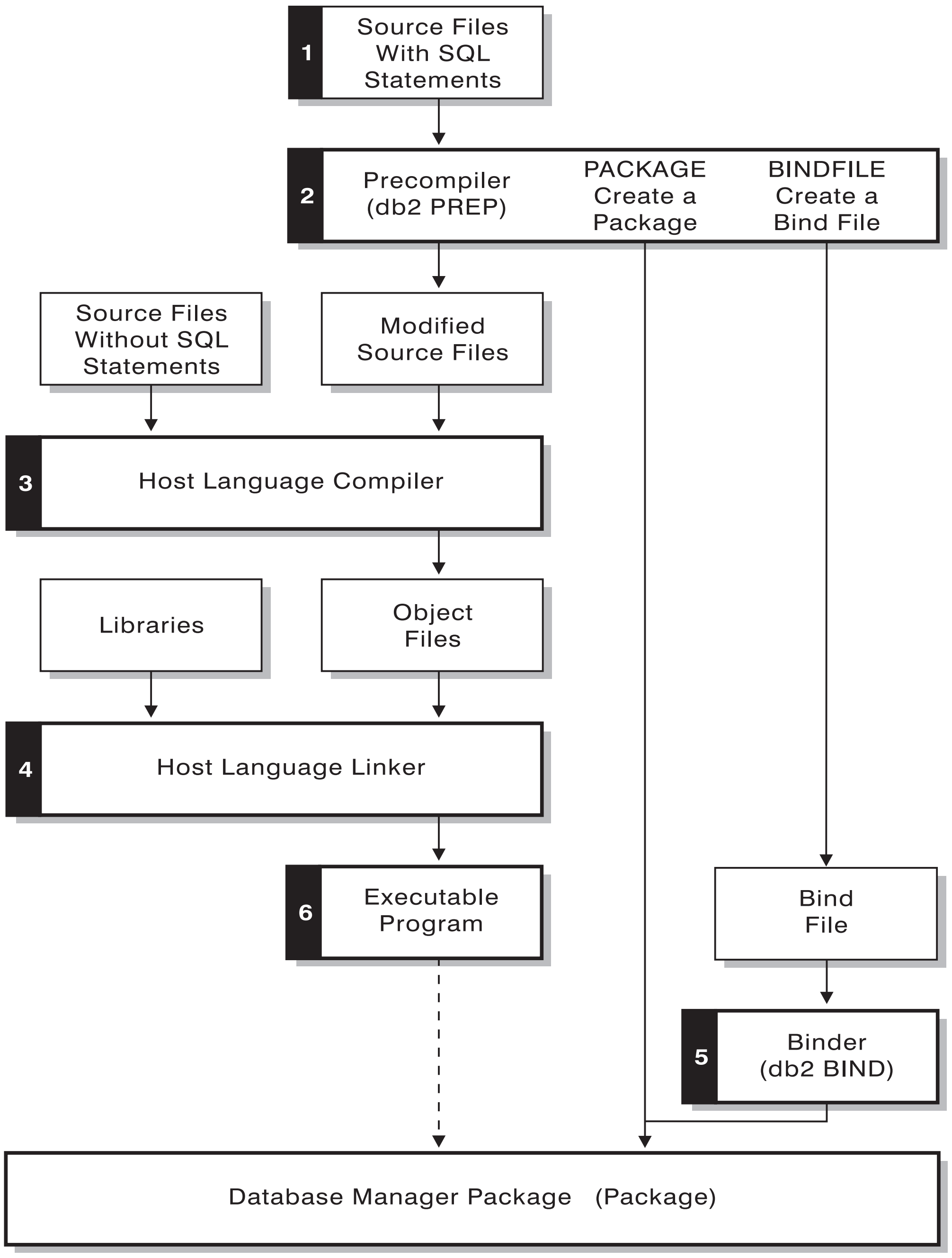


Figure 28. Preparing Programs Written in Compiled Host Languages

- Precompiling (step 2), to convert embedded SQL source statements into a form the database manager can use,

- Compiling and linking (steps 3 and 4), to create the required object modules, and,
- Binding (step 5), to create the package to be used by the database manager when the program is run.

Precompilation of Source Files Containing Embedded SQL

After you create the source files, you must precompile each host language file containing SQL statements with the PREP command for host-language source files. The precompiler converts SQL statements contained in the source file to comments, and generates the DB2 run-time API calls for those statements.

Before precompiling an application you must connect to a server, either implicitly or explicitly. Although you precompile application programs at the client workstation and the precompiler generates modified source and messages on the client, the precompiler uses the server connection to perform some of the validation.

The precompiler also creates the information the database manager needs to process the SQL statements against a database. This information is stored in a package, in a bind file, or in both, depending on the precompiler options selected.

A typical example of using the precompiler follows. To precompile a C embedded SQL source file called *filename.sqc*, you can issue the following command to create a C source file with the default name `filename.c` and a bind file with the default name `filename.bnd`:

```
DB2® PREP filename.sqc BINDFILE
```

The precompiler generates up to four types of output:

Modified Source
This file is the new version of the original source file after the precompiler converts the SQL statements into DB2 run-time API calls. It is given the appropriate host language extension.

Package
If you use the PACKAGE option (the default), or do not specify any of the BINDFILE, SYNTAX, or SQLFLAG options, the package is stored in the connected database. The package contains all the information required to execute the static SQL statements of a particular source file against this database only. Unless you specify a different name with the PACKAGE USING option, the precompiler forms the package name from the first 8 characters of the source file name.

If you use the PACKAGE option without SQLERROR CONTINUE, the database used during the precompile process must contain all of the database objects referenced by the static SQL statements in the source file. For example, you cannot precompile a SELECT statement unless the table it references exists in the database.

With the VERSION option the bindfile, (if the BINDFILE option is used), and the package (either if bound at PREP time or if a bound separately) will be designated with a particular version identifier. Many versions of packages with the same name and creator can exit at once.

Bind File
If you use the BINDFILE option, the precompiler creates a bind file (with extension `.bnd`) that contains the data required to create a package. This file can be used later with the BIND command to

bind the application to one or more databases. If you specify BINDFILE and do not specify the PACKAGE option, binding is deferred until you invoke the `BIND` command. Note that for the command line processor (CLP), the default for `PREP` does not specify the BINDFILE option. Thus, if you are using the CLP and want the binding to be deferred, you need to specify the BINDFILE option.

Specifying SQLERROR CONTINUE creates a package, even if errors occur when binding SQL statements. Those statements that fail to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to execute them at run time generates an error.

Message File If you use the MESSAGES option, the precompiler redirects messages to the indicated file. These messages include warnings and error messages that describe problems encountered during precompilation. If the source file does not precompile successfully, use the warning and error messages to determine the problem, correct the source file, and then attempt to precompile the source file again. If you do not use the MESSAGES option, precompilation messages are written to the standard output.

Precompilation of embedded SQL applications with the PRECOMPILE command

Once you have created the embedded SQL application's source files, you must precompile each host language file containing SQL statements with the `PREP` command, using the options specific to the host language. The precompiler converts SQL statements contained in the source file to comments, and generates the DB2 run-time API calls for those statements.

You must always precompile a source file against a specific database, even if eventually you do not use the database with the application. In practice, you can use a test database for development, and after you fully test the application, you can bind its bind file to one or more production databases. This practice is known as *deferred binding*.

If your application uses a code page that is not the same as your database code page, you need to consider which code page to use when precompiling.

If your application uses user-defined functions (UDFs) or user-defined distinct types (UDTs), you may need to use the FUNCPATH option when you precompile your application. This option specifies the function path that is used to resolve UDFs and UDTs for applications containing static SQL. If FUNCPATH is not specified, the default function path is *SYSIBM*, *SYSFUN*, *USER*, where *USER* refers to the current user ID.

Before precompiling an application you must connect to a server, either implicitly or explicitly. Although you precompile application programs at the client workstation and the precompiler generates modified source and messages on the client, the precompiler uses the server connection to perform some of the validation.

The precompiler also creates the information the database manager needs to process the SQL statements against a database. This information is stored in a package, in a bind file, or in both, depending on the precompiler options selected.

A typical example of using the precompiler follows. To precompile a C embedded SQL source file called *filename.sqc*, you can issue the following command to create a C source file with the default name `filename.c` and a bind file with the default name `filename.bnd`:

```
DB2 PREP filename.sqc BINDFILE
```

The precompiler generates up to four types of output:

Modified Source
This file is the new version of the original source file after the precompiler converts the SQL statements into DB2 run-time API calls. It is given the appropriate host language extension.

Package
If you use the PACKAGE option (the default), or do not specify any of the BINDFILE, SYNTAX, or SQLFLAG options, the package is stored in the connected database. The package contains all the information required to execute the static SQL statements of a particular source file against this database only. Unless you specify a different name with the PACKAGE USING option, the precompiler forms the package name from the first 8 characters of the source file name.

If you use the PACKAGE option without SQLERROR CONTINUE, the database used during the precompile process must contain all of the database objects referenced by the static SQL statements in the source file. For example, you cannot precompile a SELECT statement unless the table it references exists in the database.

With the VERSION option, the bindfile (if the BINDFILE option is used) and the package (either if bound at PREP time or if bound separately) will be designated with a particular version identifier. Many versions of packages with the same name and creator can exist at once.

Bind File
If you use the BINDFILE option, the precompiler creates a bind file (with extension `.bnd`) that contains the data required to create a package. This file can be used later with the `BIND` command to bind the application to one or more databases. If you specify BINDFILE and do not specify the PACKAGE option, binding is deferred until you invoke the `BIND` command. Note that for the command line processor (CLP), the default for `PREP` does not specify the BINDFILE option. Thus, if you are using the CLP and want the binding to be deferred, you need to specify the BINDFILE option.

Specifying SQLERROR CONTINUE creates a package, even if errors occur when binding SQL statements. Those statements that fail to bind for authorization or existence reasons can be incrementally bound at execution time if VALIDATE RUN is also specified. Any attempt to execute them at run time generates an error.

Message File
If you use the MESSAGES option, the precompiler redirects messages to the indicated file. These messages include warning and error messages that describe problems encountered during

precompilation. If the source file does not precompile successfully, use the warning and error messages to determine the problem, correct the source file, and then attempt to precompile the source file again. If you do not use the MESSAGES option, precompilation messages are written to the standard output.

Related concepts:
- "Advantages of deferred binding" in *Developing Embedded SQL Applications*
- "Character conversion between different code pages" in *Developing SQL and External Routines*
- "Character substitutions during code page conversions" in *Developing SQL and External Routines*
- "Code page conversion expansion factor" in *Developing SQL and External Routines*
- "Supported code page conversions" in *Developing SQL and External Routines*
- "When code page conversion occurs" in *Developing SQL and External Routines*
- "Host Variables in embedded SQL applications" in *Developing Embedded SQL Applications*
- "Comments in embedded SQL applications" in *Developing Embedded SQL Applications*
- "Binding embedded SQL packages to a database" in *Developing Embedded SQL Applications*
- "Connecting to DB2 databases in embedded SQL applications" on page 1290

Related tasks:
- "Declaring host variables in embedded SQL applications" in *Developing Embedded SQL Applications*
- "Referencing host variables in embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:
- "BIND" on page 441
- "PRECOMPILE " on page 635

Compiling and linking source files containing embedded SQL

When precompiling embedded SQL source files, the `PRECOMPILE` command generates modified source files with a file extension applicable to the programming language.

Compile the modified source files (and any additional source files that do not contain SQL statements) using the appropriate host language compiler. The language compiler converts each modified source file into an *object module*.

Refer to the programming documentation for your operating platform for any exceptions to the default compiler options. Refer to your compiler's documentation for a complete description of available compiler options.

The host language linker creates an executable application. For example:
- On Windows operating systems, the application can be an executable file or a dynamic link library (DLL).
- On UNIX and Linux based operating systems, the application can be an executable load module or a shared library.

Note: Although applications can be DLLs on Windows operating systems, the DLLs are loaded directly by the application and not by the DB2 database manager. On Windows operating systems, the database manager loads embedded SQL stored procedures and user-defined functions as DLLs.

To create the executable file, link the following:
- User object modules, generated by the language compiler from the modified source files and other files not containing SQL statements.
- Host language library APIs, supplied with the language compiler.
- The database manager library containing the database manager APIs for your operating environment. Refer to the appropriate programming documentation for your operating platform for the specific name of the database manager library you need for your database manager APIs.

Related tasks:
- "Building and running embedded SQL applications written in REXX" in *Developing Embedded SQL Applications*
- "Building applications in C or C++ using the sample build script (UNIX)" in *Developing Embedded SQL Applications*
- "Building IBM COBOL applications on AIX" in *Developing Embedded SQL Applications*
- "Building UNIX Micro Focus COBOL applications" in *Developing Embedded SQL Applications*

Package recreation using the BIND command and an existing bind file

Binding is the process that creates the package the database manager needs to access the database when the application is executed. By default the PRECOMPILE command creates a package. Binding is done implicitly at precompile time unless the BINDFILE option is specified. The PACKAGE option allows you to specify a package name for the package created at precompile time.

A typical example of using the `BIND` command follows. To bind a bind file named *filename.bnd* to the database, you can issue the following command:

```
BIND filename.bnd
```

One package is created for each separately precompiled source code module. If an application has five source files, of which three require precompilation, three packages or bind files are created. By default, each package is given a name that is the same as the name of the source module from which the `.bnd` file originated, but truncated to 8 characters. To explicitly specify a different package name, you must use the PACKAGE USING option on the `PREP` command. The version of a package is given by the VERSION precompile option and defaults to the empty string. If the name and schema of this newly created package is the same as a package that currently exists in the target database, but the version identifier differs, a new package is created and the previous package still remains. However if a package exists that matches the name, schema and the version of the package being bound, then that package is dropped and replaced with the new package being bound (specifying ACTION ADD on the bind would prevent that and an error (SQL0719) would be returned instead).

Related concepts:

- “Precompilation of embedded SQL applications with the PRECOMPILE command” on page 1299
- “Precompiler generated timestamps” in *Developing Embedded SQL Applications*

Related reference:

- “BIND” on page 441
- “PRECOMPILE ” on page 635

Generation of sequential values

Generating sequential values is a common database application development problem. The best solution to that problem is to use sequence objects and sequence expressions in SQL. Each *sequence object* is a uniquely named database object that can be accessed only by sequence expressions. There are two *sequence expressions*: the PREVVAL expression and the NEXTVAL expression. The PREVVAL expression returns the value most recently generated in the application process for the specified sequence object. Any NEXTVAL expressions occuring in the same statement as the PREVVAL expression have no effect on the value generated by the PREVAL expression in that statement. The NEXTVAL sequence expression increments the value of the sequence object and returns the new value of the sequence object.

To create a sequence object, issue the CREATE SEQUENCE statement. For example, to create a sequence object called id_values using the default attributes, issue the following statement:

```
  CREATE SEQUENCE id_values
```

To generate the first value in the application session for the sequence object, issue a VALUES statement using the NEXTVAL expression:

```
VALUES NEXTVAL FOR id_values

1
-----------
          1

            1 record(s) selected.
```

To display the current value of the sequence object, issue a VALUES statement using the PREVVAL expression:

```
VALUES PREVVAL FOR id_values

1
-----------
          1

            1 record(s) selected.
```

You can repeatedly retrieve the current value of the sequence object, and the value that the sequence object returns does not change until you issue a NEXTVAL expression. In the following example, the PREVVAL expression returns a value of 1, until the NEXTVAL expression in the current connection increments the value of the sequence object:

```
VALUES PREVVAL FOR id_values

1
-----------
          1
```

```
            1 record(s) selected.

VALUES PREVVAL FOR id_values

1
-----------
          1

            1 record(s) selected.

VALUES NEXTVAL FOR id_values

1
-----------
          2

            1 record(s) selected.

VALUES PREVVAL FOR id_values

1
-----------
          2

            1 record(s) selected.
```

To update the value of a column with the next value of the sequence object, include the NEXTVAL expression in the UPDATE statement, as follows:

```
UPDATE staff
  SET id = NEXTVAL FOR id_values
  WHERE id = 350
```

To insert a new row into a table using the next value of the sequence object, include the NEXTVAL expression in the INSERT statement, as follows:

```
INSERT INTO staff (id, name, dept, job)
  VALUES (NEXTVAL FOR id_values, 'Kandil', 51, 'Mgr')
```

Related reference:

- "CREATE SEQUENCE statement" in *SQL Reference, Volume 2*

Related samples:

- "DbSeq.java -- How to create, alter and drop a sequence in a database (JDBC)"

Management of sequence behavior

You can tailor the behavior of sequence objects to meet the needs of your application. You change change the attributes of a sequence object when you issue the CREATE SEQUENCE statement to create a new sequence object, and when you issue the ALTER SEQUENCE statement for an existing sequence object. Following are some of the attributes of a sequence object that you can specify:

Data type
: The AS clause of the CREATE SEQUENCE statement specifies the numeric data type of the sequence object. The data type determines the possible minimum and maximum values of the sequence object (the minimum and maximum values for a data type are listed in the topic describing SQL limits). You cannot change the data type of a sequence object; instead, you must drop the sequence object by issuing the DROP SEQUENCE statement and issue a CREATE SEQUENCE statement with the new data type.

Start value
The START WITH clause of the CREATE SEQUENCE statement sets the initial value of the sequence object. The RESTART WITH clause of the ALTER SEQUENCE statement resets the value of the sequence object to a specified value.

Minimum value
The MINVALUE clause sets the minimum value of the sequence object.

Maximum value
The MAXVALUE clause sets the maximum value of the sequence object.

Increment value
The INCREMENT BY clause sets the value that each NEXTVAL expression adds to the current value of the sequence object. To decrement the value of the sequence object, specify a negative value.

Sequence cycling
The CYCLE clause causes the value of a sequence object that reaches its maximum or minimum value to generate its respective minimum value or maximum value on the following NEXTVAL expression.

For example, to create a sequence object called id_values that starts with a minimum value of 0, has a maximum value of 1000, increments by 2 with each NEXTVAL expression, and returns to its minimum value when the maximum value is reached, issue the following statement:

```
CREATE SEQUENCE id_values
  START WITH 0
  INCREMENT BY 2
  MAXVALUE 1000
  CYCLE
```

Related reference:
- "ALTER SEQUENCE statement" in *SQL Reference, Volume 2*
- "CREATE SEQUENCE statement" in *SQL Reference, Volume 2*
- "SQL and XQuery limits" in *SQL Reference, Volume 1*

Sequence objects compared to identity columns

Although sequence objects and identity columns appear to serve similar purposes for DB2 applications, there is an important difference. An identity column automatically generates values for a column in a single table. A sequence object generates sequential values upon request that can be used in any SQL statement.

Authorization Considerations for Embedded SQL

An *authorization* allows a user or group to perform a general task such as connecting to a database, creating tables, or administering a system. A *privilege* gives a user or group the right to access one specific database object in a specified way. DB2® uses a set of privileges to provide protection for the information that you store in it.

Most SQL statements require some type of privilege on the database objects which the statement utilizes. Most API calls usually do not require any privilege on the database objects which the call utilizes, however, many APIs require that you possess the necessary authority in order to invoke them. The DB2 APIs enable you to perform the DB2 administrative functions from within your application program. For example, to recreate a package stored in the database without the need for a bind file, you can use the `sqlarbnd` (or REBIND) API.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. Group membership is considered for the execution of dynamic SQL statements, but not for static SQL statements. `PUBLIC` privileges are, however, considered for the execution of static SQL statements. For example, suppose you have an embedded SQL stored procedure with statically bound SQL queries against a table called `STAFF`. If you try to build this procedure with the `CREATE PROCEDURE` statement, and your account belongs to a group that has the select privilege for the `STAFF` table, the `CREATE` statement will fail with a SQL0551N error. For the `CREATE` statement to work, your account directly needs the select privilege on the `STAFF` table.

When you design your application, consider the privileges your users will need to run the application. The privileges required by your users depend on:

- Whether your application uses dynamic SQL, including JDBC and DB2 CLI, or static SQL. For information about the privileges required to issue a statement, see the description of that statement.
- Which APIs the application uses. For information about the privileges and authorities required for an API call, see the description of that API.

Groups provide a convenient means of performing authorization for a collection of users without having to grant or revoke privileges for each user individually. In general, group membership is considered for dynamic SQL statements, but is not considered for static SQL statements. The exception to this general case occurs when privileges are granted to PUBLIC: these are considered when static SQL statements are processed.

Consider two users, PAYROLL and BUDGET, who need to perform queries against the STAFF table. PAYROLL is responsible for paying the employees of the company, so it needs to issue a variety of SELECT statements when issuing paychecks. PAYROLL needs to be able to access each employee's salary. BUDGET is responsible for determining how much money is needed to pay the salaries. BUDGET should not, however, be able to see any particular employee's salary.

Because PAYROLL issues many different SELECT statements, the application you design for PAYROLL could probably make good use of dynamic SQL. The dynamic SQL would require that PAYROLL have SELECT privilege on the STAFF table. This requirement is not a problem because PAYROLL requires full access to the table.

BUDGET, on the other hand, should not have access to each employee's salary. This means that you should not grant SELECT privilege on the STAFF table to BUDGET. Because BUDGET does need access to the total of all the salaries in the STAFF table, you could build a static SQL application to execute a SELECT SUM(SALARY) FROM STAFF, bind the application and grant the EXECUTE privilege on your application's package to BUDGET. This enables BUDGET to obtain the required information, without exposing the information that BUDGET should not see.

Related concepts:

- "Authorization" on page 62
- "Authorization considerations for dynamic SQL" on page 1307
- "Authorization considerations for static SQL" on page 1307

Authorization considerations for dynamic SQL

To use dynamic SQL in a package bound with DYNAMICRULES RUN (default), the person who runs a dynamic SQL application must have the privileges necessary to issue each SQL request performed, as well as the EXECUTE privilege on the package. The privileges can be granted to the user's authorization ID, to any group of which the user is a member, or to PUBLIC.

If you bind the application with the DYNAMICRULES BIND option, DB2 associates your authorization ID with the application packages. This allows any user who runs the application to inherit the privileges associated with your authorization ID.

If the program contains no static SQL, the person binding the application (for embedded dynamic SQL applications) only needs the BINDADD authority on the database. Again, this privilege can be granted to the user's authorization ID, to a group of which the user is a member, or to PUBLIC.

When a package exhibits bind or define behavior, the user that runs the application needs only the EXECUTE privilege on the package to run it. At run-time, the binder of a package that exhibits bind behavior must have the privileges necessary to execute all the dynamic statements generated by the package, because all authorization checking for dynamic statements is done using the ID of the binder and not the executors. Similarly, the definer of a routine whose package exhibits define behavior must have all the privileges necessary to execute all the dynamic statements generated by the define behavior package. If you have SYSADM or DBADM authority and create a bind behavior package, consider using the OWNER BIND option to designate a different authorization ID. The OWNER BIND option prevents a package from automatically inheriting SYSADM or DBADM privileges within dynamic SQL statements. For more information on the DYNAMICRULES and OWNER bind options, refer to the BIND command. For more information on package behaviors, see the description of DYNAMICRULES effects on dynamic SQL statements.

Related concepts:
- "Authorization Considerations for Embedded SQL" on page 1305
- "Authorization considerations for static SQL" on page 1307
- "Authorizations and binding of routines that contain SQL" on page 83

Related reference:
- "BIND" on page 441

Authorization considerations for static SQL

To use static SQL, the user running the application only needs the EXECUTE privilege on the package. No privileges are required for each of the statements that make up the package. The EXECUTE privilege can be granted to the user's authorization ID, to any group of which the user is a member, or to PUBLIC.

Unless you specify the VALIDATE RUN option when binding the application, the authorization ID you use to bind the application must have the privileges necessary to perform all the statements in the application. If VALIDATE RUN was specified at BIND time, all authorization failures for any static SQL within this package will not cause the BIND to fail and those statements will be revalidated at run time. The person binding the application must always have BINDADD authority. The privileges needed to execute the statements must be granted to the user's authorization ID or to PUBLIC. Group privileges are not used when binding static SQL statements. As with dynamic SQL, the BINDADD privilege can be granted to the user authorization ID, to a group of which the user is a member, or to PUBLIC.

These properties of static SQL give you very precise control over access to information in DB2.

Related concepts:
- "Authorization considerations for dynamic SQL" on page 1307
- "Authorization Considerations for Embedded SQL" on page 1305

Related reference:
- "BIND" on page 441

Effect of DYNAMICRULES bind option on dynamic SQL

The PRECOMPILE command and BIND command option DYNAMICRULES determines what values apply at run-time for the following dynamic SQL attributes:
- The authorization ID that is used during authorization checking.
- The qualifier that is used for qualification of unqualified objects.
- Whether the package can be used to dynamically prepare the following statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY and SET EVENT MONITOR STATE statements.

In addition to the DYNAMICRULES value, the run-time environment of a package controls how dynamic SQL statements behave at run-time. The two possible run-time environments are:
- The package runs as part of a stand-alone program
- The package runs within a routine context

The combination of the DYNAMICRULES value and the run-time environment determine the values for the dynamic SQL attributes. That set of attribute values is called the dynamic SQL statement behavior. The four behaviors are:

Run behavior DB2 uses the authorization ID of the user (the ID that initially connected to DB2) executing the package as the value to be used for authorization checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

Bind behavior At run-time, DB2 uses all the rules that apply to static SQL for authorization and qualification. That is, take the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

Define behavior
Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES DEFINEBIND or DYNAMICRULES

DEFINERUN. DB2 uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

Invoke behavior

Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES INVOKEBIND or DYNAMICRULES INVOKERUN. DB2 uses the current statement authorization ID in effect when the routine is invoked as the value to be used for authorization checking of dynamic SQL and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table:

Invoking Environment	**ID Used**
Any static SQL	Implicit or explicit value of the OWNER of the package the SQL invoking the routine came from.
Used in definition of view or trigger	Definer of the view or trigger.
Dynamic SQL from a run behavior package	ID used to make the initial connection to DB2.
Dynamic SQL from a define behavior package	Definer of the routine that uses the package that the SQL invoking the routine came from.
Dynamic SQL from an invoke behavior package	Current authorization ID invoking the routine.

The following table shows the combination of the DYNAMICRULES value and the run-time environment that yields each dynamic SQL behavior.

Table 147. How DYNAMICRULES and the Run-Time Environment Determine Dynamic SQL Statement Behavior

DYNAMICRULES Value	**Behavior of Dynamic SQL Statements in a Standalone Program Environment**	**Behavior of Dynamic SQL Statements in a Routine Environment**
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior
INVOKERUN	Run behavior	Invoke behavior

The following table shows the dynamic SQL attribute values for each type of dynamic SQL behavior.

Table 148. Definitions of Dynamic SQL Statement Behaviors

Dynamic SQL Attribute	Setting for Dynamic SQL Attributes: Bind Behavior	Setting for Dynamic SQL Attributes: Run Behavior	Setting for Dynamic SQL Attributes: Define Behavior	Setting for Dynamic SQL Attributes: Invoke Behavior
Authorization ID	The implicit or explicit value of the OWNER BIND option	ID of User Executing Package	Routine definer (not the routine's package owner)	Current statement authorization ID when routine is invoked.
Default qualifier for unqualified objects	The implicit or explicit value of the QUALIFIER BIND option	CURRENT SCHEMA Special Register	Routine definer (not the routine's package owner)	Current statement authorization ID when routine is invoked.
Can execute GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT ON, RENAME, SET INTEGRITY and SET EVENT MONITOR STATE	No	Yes	No	No

Related concepts:

- "Authorization considerations for dynamic SQL" on page 1307
- "Authorizations and binding of routines that contain SQL" on page 83
- "Precompilation of embedded SQL applications with the PRECOMPILE command" on page 1299

Related tasks:

- "Setting up the embedded SQL development environment" in *Developing Embedded SQL Applications*

Compound SQL guidelines

To reduce database manager overhead, you can group several SQL statements into a single executable block. Because the SQL statements in the block are substatements that could be executed individually, this kind of code is called *compound SQL*. In addition to reducing database manager overhead, compound SQL reduces the number of requests that have to be transmitted across the network for remote clients.

There are two types of compound SQL:

- **Atomic**

  The application receives a response from the database manager when all substatements have completed successfully or when one substatement ends in an error. If one substatement ends in an error, the entire block is considered to have ended in an error. Any changes made to the database within the block are rolled back.

  Atomic compound SQL is not supported with DB2 Connect
- **Not Atomic**

  The application receives a response from the database manager when all substatements have completed. All substatements within a block are executed regardless of whether or not the preceding substatement completed successfully. The group of statements can only be rolled back if the unit of work containing the NOT ATOMIC compound SQL is rolled back.

Compound SQL is supported in stored procedures, which are also known as DARI routines, and in the following application development processes:
- Embedded static SQL
- DB2 Call Level Interface
- JDBC

Dynamic Compound SQL Statements

Dynamic compound statements are compiled by DB2 as a single statement. This statement can be used effectively for short scripts that require little control flow logic but significant data flow. For larger constructs with nested complex control flow, consider using SQL procedures.

In a dynamic compound statement you can use the following elements in declarations:
- SQL variables in variable declarations of substatements
- Conditions in the substatements based on the SQLSTATE values of the condition declaration
- One or more SQL procedural statements

Dynamic compound statements can also use several flow logic statements, such as the FOR statement, the IF statement, the ITERATE statement, and the WHILE statement.

If an error occurs in a dynamic compound statement, all prior SQL statements are rolled back and the remaining SQL statements in the dynamic compound statement are not processed.

A dynamic compound statement can be embedded in a trigger, SQL function, or SQL method, or issued through dynamic SQL statements. This executable statement can be dynamically prepared. No privileges are required to invoke the statement but the authorization ID associated with the statement must have the necessary privileges to invoke the SQL statements in the compound statement.

Related concepts:
- "Query tuning guidelines" in *Performance Guide*

Units of work and transactions

Units of work

A transaction is commonly referred to in DB2 Database for Linux, UNIX, and Windows as a *unit of work*. A unit of work is a recoverable sequence of operations within an application process. It is used by the database manager to ensure that a database is in a consistent state. Any reading from or writing to the database is done within a unit of work.

For example, a bank transaction might involve the transfer of funds from a savings account to a checking account. After the application subtracts an amount from the savings account, the two accounts are inconsistent, and remain so until the amount is added to the checking account. When *both* steps are completed, a point of consistency is reached. The changes can be committed and made available to other applications.

A unit of work is started implicitly when the first SQL statement is issued against the database. All subsequent reads and writes by the same application are considered part of the same unit of work. The application must end the unit of work by issuing either a COMMIT or a ROLLBACK statement. The COMMIT statement makes permanent all changes made within a unit of work. The ROLLBACK statement removes these changes from the database. If the application ends normally without either of these statements being explicitly issued, the unit of work is automatically committed. If it ends abnormally in the middle of a unit of work, the unit of work is automatically rolled back. Once issued, a COMMIT or a ROLLBACK cannot be stopped. With some multi-threaded applications, or some operating systems (such as Windows), if the application ends normally without either of these statements being explicitly issued, the unit of work is automatically rolled back. It is recommended that your applications always explicitly commit or roll back complete units of work. If part of a unit of work does not complete successfully, the updates are rolled back, leaving the participating tables as they were before the transaction began. This ensures that requests are neither lost nor duplicated.

There is no physical representation of a unit of work because it is a series of instructions (SQL statements).

Related reference:
- "COMMIT " on page 1157
- "ROLLBACK " on page 1179

Remote unit of work

A *remote unit of work* lets a user or application program read or update data at one location per unit of work. It supports access to one database within a unit of work. While an application program can update several remote databases, it can only access one database within a unit of work.

Remote unit of work has the following characteristics:
- Multiple requests (SQL statements) per unit of work are supported.
- Multiple cursors per unit of work are supported.
- Each unit of work can update only one database.
- The application program either commits or rolls back the unit of work. In certain error circumstances, the database server or DB2 Connect might roll back the unit of work.

For example, Figure 29 on page 1313 shows a database client running a funds transfer application that accesses a database containing checking and savings account tables, as well as a transaction fee schedule. The application must:
- Accept the amount to transfer from the user interface.
- Subtract the amount from the savings account, and determine the new balance.
- Read the fee schedule to determine the transaction fee for a savings account with the given balance.
- Subtract the transaction fee from the savings account.

- Add the amount of the transfer to the checking account.
- Commit the transaction (unit of work).

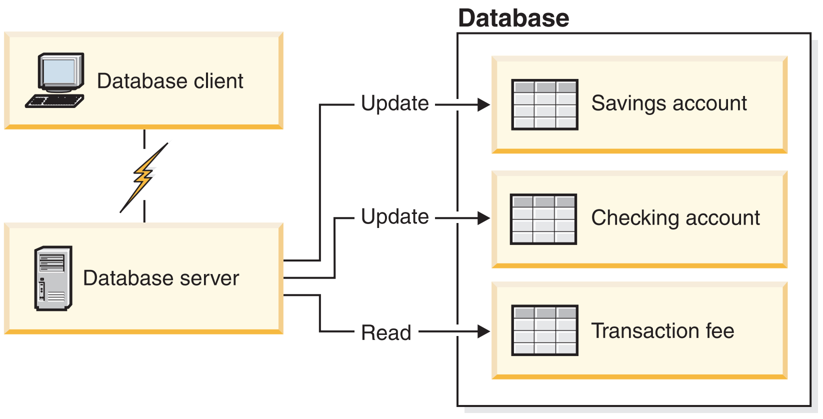


Figure 29. Using a Single Database in a Transaction

To set up such an application, you must:

1. Create the tables for the savings account, checking account and transaction fee schedule in the same database.
2. If physically remote, set up the database server to use the appropriate communications protocol.
3. If physically remote, catalog the node and the database to identify the database on the database server.
4. Precompile your application program to specify a type 1 connection; that is, specify CONNECT(1) on the PREP command.

Related concepts:

- "DB2 Connect and DRDA" in *DB2 Connect User's Guide*
- "Distributed requests" in *DB2 Connect User's Guide*
- "Distributed Relational Database Architecture" in *DB2 Connect User's Guide*
- "Remote units of work" in *Developing SQL and External Routines*

Concurrent transactions and multi-threaded database access in embedded SQL applications

One feature of some operating systems is the ability to run several threads of execution within a single process. The multiple threads allow an application to handle asynchronous events, and makes it easier to create event-driven applications, without resorting to polling schemes. The information that follows describes how the DB2 database manager works with multiple threads, and lists some design guidelines that you should keep in mind.

If you are not familiar with terms relating to the development of multi-threaded applications (such as critical section and semaphore), consult the programming documentation for your operating system.

A DB2 embedded SQL application can execute SQL statements from multiple threads using *contexts*. A context is the environment from which an application runs all SQL statements and API calls. All connections, units of work, and other database resources are associated with a specific context. Each context is associated with one or more threads within an application. Developing multi-threaded embedded SQL applications with thread-safe code is only supported in C and C++. It is possible to write your own precompiler, that along with features supplied by the language allows concurrent multithread database access.

For each executable SQL statement in a context, the first run-time services call always tries to obtain a latch. If it is successful, it continues processing. If not (because an SQL statement in another thread of the same context already has the latch), the call is blocked on a signaling semaphore until that semaphore is posted, at which point the call gets the latch and continues processing. The latch is held until the SQL statement has completed processing, at which time it is released by the last run-time services call that was generated for that particular SQL statement.

The net result is that each SQL statement within a context is executed as an atomic unit, even though other threads may also be trying to execute SQL statements at the same time. This action ensures that internal data structures are not altered by different threads at the same time. APIs also use the latch used by run-time services; therefore, APIs have the same restrictions as run-time services routines within each context.

Contexts may be exchanged between threads in a process, but not exchanged between processes. One use of multiple contexts is to provide support for concurrent transactions.

In the default implementation of threaded applications against a DB2 database, serialization of access to the database is enforced by the database APIs. If one thread performs a database call, calls made by other threads will be blocked until the first call completes, even if the subsequent calls access database objects that are unrelated to the first call. In addition, all threads within a process share a commit scope. True concurrent access to a database can only be achieved through separate processes, or by using the APIs that are described in this topic.

DB2 database systems provide APIs that can be used to allocate and manipulate separate environments (contexts) for the use of database APIs and embedded SQL. Each context is a separate entity, and any connection or attachment using one context is independent of all other contexts (and thus all other connections or attachments within a process). In order for work to be done on a context, it must first be associated with a thread. A thread must always have a context when making database API calls or when using embedded SQL.

All DB2 database system applications are multithreaded by default, and are capable of using multiple contexts. You can use the following DB2 APIs to use multiple contexts. Specifically, your application can create a context for a thread, attach to or detach from a separate context for each thread, and pass contexts between threads. If your application does not call *any* of these APIs, DB2 will automatically manage the multiple contexts for your application:

- `sqleAttachToCtx` - Attach to context
- `sqleBeginCtx` - Create and attach to an application context
- `sqleDetachFromCtx` - Detach from context
- `sqleEndCtx` - Detach and destory application context
- `sqleGetCurrentCtx` - Get current context
- `sqleInterruptCtx` - Interrupt context

These APIs have no effect (that is, they are no-ops) on platforms that do not support application threading.

Contexts need not be associated with a given thread for the duration of a connection or attachment. One thread can attach to a context, connect to a database, detach from the context, and then a second thread can attach to the context and continue doing work using the already existing database connection. Contexts can be passed around among threads in a process, but not among processes.

Even if the new APIs are used, the following APIs continue to be serialized:
- `sqlabndx` - Bind
- `sqlaprep` - Precompile Program
- `sqluexpr` - Export
- `db2Import` and `sqluimpr` - Import

Notes:
1. The DB2 CLI automatically uses multiple contexts to achieve thread-safe, concurrent database access on platforms that support multi-threading. While not recommended by DB2, users can explicitly disable this feature if required.
2. By default, AIX does not permit 32-bit applications to attach to more than 11 shared memory segments per process, of which a maximum of 10 can be used for DB2 connections.

   When this limit is reached, DB2 returns SQLCODE -1224 on an SQL CONNECT. DB2 Connect also has the 10-connection limitation if local users are running two-phase commit over SNA, or two-phase commit with a TP Monitor (SNA or TCP/IP).

   The AIX environment variable `EXTSHM` can be used to increase the maximum number of shared memory segments to which a process can attach.

   To use `EXTSHM` with DB2, do the following:

   In client sessions:
   ```
   export EXTSHM=ON
   ```
   When starting the DB2 server:
   ```
   export EXTSHM=ON
   db2set DB2ENVLIST=EXTSHM
   db2start
   ```
   On DPF, also add the following lines to your userprofile or usercshrc files:
   ```
   EXTSHM=ON
   export EXTSHM
   ```
   An alternative is to move the local database or DB2 Connect into another machine and to access it remotely, or to access the local database or the DB2 Connect database with TCP/IP loop-back by cataloging it as a remote node that has the TCP/IP address of the local machine.

Related concepts:
- "Concurrent transactions" in *Developing SQL and External Routines*
- "Recommendations for using multiple threads" in *Developing Embedded SQL Applications*
- "Executing SQL statements in embedded SQL applications" in *Developing Embedded SQL Applications*

Related reference:

- “Precompiler customization APIs” in *Administrative API Reference*
- “sqleAttachToCtx API - Attach to context” in *Administrative API Reference*
- “sqleBeginCtx API - Create and attach to an application context” in *Administrative API Reference*
- “sqleDetachFromCtx API - Detach from context” in *Administrative API Reference*
- “sqleEndCtx API - Detach from and free the memory associated with an application context” in *Administrative API Reference*
- “sqleGetCurrentCtx API - Get current context” in *Administrative API Reference*
- “sqleInterruptCtx API - Interrupt context” in *Administrative API Reference*
- “sqleSetTypeCtx API - Set application context type” in *Administrative API Reference*
- “Administrative APIs and application migration” in *Administrative API Reference*
- “Changed APIs and data structures” in *Administrative API Reference*

Related samples:
- “dbthrds.sqc -- How to use multiple context APIs on UNIX (C)”
- “dbthrds.sqC -- How to use multiple context APIs on UNIX (C++)”

Ending a transaction with the COMMIT statement

The COMMIT statement ends the current transaction and makes the database changes performed during the transaction visible to other processes.

Procedure:

Commit changes as soon as application requirements permit. In particular, write your programs so that uncommitted changes are not held while waiting for input from a terminal, as this can result in database resources being held for a long time. Holding these resources prevents other applications that need these resources from running.

Your application programs should explicitly end any transactions before terminating.

If you do not end transactions explicitly, DB2 automatically commits all the changes made during the program’s pending transaction when the program ends successfully, except on Windows operating systems. On Windows operating systems, if you do not explicitly commit the transaction, the database manager always rolls back the changes.

DB2 rolls back the changes under the following conditions:
- A log full condition
- Any other system condition that causes database manager processing to end

The COMMIT statement has no effect on the contents of host variables.

Related concepts:
- “Error information in the SQLCODE, SQLSTATE, and SQLWARN fields” in *Developing Embedded SQL Applications*
- “Implicit Ending of a Transaction” on page 1317

Related reference:

- "COMMIT " on page 1157

Implicit Ending of a Transaction

If your program terminates without ending the current transaction, DB2 implicitly ends the current transaction by issuing a ROLLBACK statement. This implicit rollback applies to all operating systems, all application terminations (normal and abnormal) and to the applications that use the DB2 context APIs for multi-threaded database access. If you want the transaction to be committed, you must explicitly issue the COMMIT statement.

Note: What your program considers to be an abnormal termination might not be considered abnormal by the database manager. For example, you might code exit(-16) when your application encounters an unexpected error and terminate your application abruptly. The database manager considers this to be a normal termination. The database manager considers items such as an exception or a segmentation violation as abnormal terminations.

Related concepts:

- "Concurrent transactions and multi-threaded database access in embedded SQL applications" on page 1313

Related reference:

- "COMMIT " on page 1157
- "ROLLBACK " on page 1179

Security and Java Applications

The sections that follow describe security considerations for SQLJ, JDBC, the Type 2 JDBC driver, and the Universal JDBC driver.

SQLJ Considerations

Controlling the execution of SQL statements in SQLJ

You can use selected methods of the SQLJ `ExecutionContext` class to control or monitor the execution of SQL statements.

To use `ExecutionContext` methods, follow these steps:

1. Acquire an *execution context*.

   There are two ways to acquire an execution context:

   - Acquire the default execution context from the connection context. For example:

     ```
     ExecutionContext execCtx = connCtx.getExecutionContext();
     ```
   - Create a new execution context by invoking the constructor for `ExecutionContext`. For example:

     ```
     ExecutionContext execCtx=new ExecutionContext();
     ```
2. Associate the execution context with an SQL statement.

   To do that, specify an execution context after the connection context in the execution clause that contains the SQL statement. For example:

   ```
   #sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
   ```
3. Invoke `ExecutionContext` methods.

Some ExecutionContext methods are applicable before the associated SQL statement is executed, and some are applicable only after their associated SQL statement is executed.

For example, you can use method `getUpdateCount` to count the number of rows that are deleted by a DELETE statement after you execute the DELETE statement:

```
#sql [connCtx, execCtx] {DELETE FROM EMPLOYEE WHERE SALARY > 10000};
System.out.println("Deleted " + execCtx.getUpdateCount() + " rows");
```

SQLJ SET-TRANSACTION-clause

The SET TRANSACTION clause sets the isolation level for the current unit of work.

Syntax:

```
►►──SET TRANSACTION──ISOLATION LEVEL──┬─READ COMMITTED───┬──────────────────────────────────►◄
                                      ├─READ UNCOMMITTED─┤
                                      ├─REPEATABLE READ──┤
                                      └─SERIALIZABLE─────┘
```

Description:

ISOLATION LEVEL
Specifies one of the following isolation levels:

READ COMMITTED
Specifies that the current DB2 isolation level is cursor stability.

READ UNCOMMITTED
Specifies that the current DB2 isolation level is uncommitted read.

REPEATABLE READ
Specifies that the current DB2 isolation level is read stability.

SERIALIZABLE
Specifies that the current DB2 isolation level is repeatable read.

Usage notes:

You can execute SET TRANSACTION only at the beginning of a transaction.

Setting the isolation level for an SQLJ transaction

To set the isolation level for a unit of work within an SQLJ program, use the SET TRANSACTION ISOLATION LEVEL clause. Table 149 shows the values that you can specify in the SET TRANSACTION ISOLATION LEVEL clause and their DB2 equivalents.

Table 149. Equivalent SQLJ and DB2 isolation levels

SET TRANSACTION value	DB2 isolation level
SERIALIZABLE	Repeatable read
REPEATABLE READ	Read stability
READ COMMITTED	Cursor stability
READ UNCOMMITTED	Uncommitted read

The isolation level affects the underlying JDBC connection as well as the SQLJ connection.

Related concepts:
- "Isolation levels" in *SQL Reference, Volume 1*

SQLJ context-clause

A context clause specifies a connection context, an execution context, or both. You use a connection context to connect to a data source. You use an execution context to monitor and modify SQL statement execution.

Syntax:

```
►►──[──┬─connection-context─────────────────────┬─]──────────────────────────────────────────►◄
       ├─execution-context──────────────────────┤
       └─connection-context──,──execution context─┘
```

Description:

connection-context
: Specifies a valid Java™ identifier that is declared earlier in the SQLJ program. That identifier must be declared as an instance of the connection context class that SQLJ generates for a connection declaration clause.

execution-context
: Specifies a valid Java identifier that is declared earlier in the SQLJ program. That identifier must be declared as an instance of class `sqlj.runtime.ExecutionContext`.

Usage notes:
- If you do not specify a connection context in an executable clause, SQLJ uses the default connection context.
- If you do not specify an execution context, SQLJ obtains the execution context from the connection context of the statement.

Related tasks:
- "Connecting to a data source using SQLJ" on page 1319
- "Controlling the execution of SQL statements in SQLJ" on page 1317

Connecting to a data source using SQLJ

In an SQLJ application, as in any other DB2 application, you must be connected to a database server before you can execute SQL statements. In SQLJ, as in JDBC, a database server is called a *data source*.

You can use one of the following techniques to connect to a data source.

Connection technique 1: This technique uses the JDBC `DriverManager` as the underlying means for creating the connection. Use it with any level of the JDBC driver.

1. Execute an SQLJ *connection declaration clause*.

   Doing this generates a *connection context class*. The simplest form of the connection declaration clause is:

   ```
   #sql context context-class-name;
   ```

The name of the generated connection context class is *context-class-name*.

2. Load a JDBC driver by invoking the `Class.forName` method:
   - For the IBM DB2 Driver for JDBC and SQLJ, invoke `Class.forName` this way:
     ```
     Class.forName("com.ibm.db2.jcc.DB2Driver");
     ```
   - For the DB2 JDBC Type 2 Driver, invoke `Class.forName` this way:
     ```
     Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
     ```
3. Invoke the constructor for the connection context class that you created in step 1 on page 1319.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in one of the following forms:
   ```
   connection-context-class connection-context-object=
     new connection-context-class(String url, boolean autocommit);

   connection-context-class connection-context-object=
     new connection-context-class(String url, String user,
       String password, boolean autocommit);
   connection-context-class connection-context-object=
     new connection-context-class(String url, Properties info,
       boolean autocommit);
   ```
   The meanings of the parameters are:

   url A string that specifies the location name that is associated with the data source. That argument has one of the forms that are specified in Connect to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ. The form depends on which JDBC driver you are using.

   user **and** *password*
   Specify a user ID and password for connection to the data source, if the data source to which you are connecting requires them.

   info
   Specifies an object of type `java.util.Properties` that contains a set of driver properties for the connection. For the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (DB2 JDBC Type 2 Driver), you should specify only the `user` and `password` properties. For the IBM DB2 Driver for JDBC and SQLJ, you can specify any of the properties listed in Properties for the IBM DB2 Driver for JDBC and SQLJ.

   autocommit
   Specifies whether you want the database manager to issue a COMMIT after every statement. Possible values are `true` or `false`. If you specify `false`, you need to do explicit commit operations.

The following code uses connection technique 1 to create a connection to location NEWYORK. The connection requires a user ID and password, and does not require autocommit. The numbers to the right of selected statements correspond to the previously-described steps.

```
#sql context Ctx;            // Create connection context class Ctx       1
String userid="dbadm";       // Declare variables for user ID and password
String password="dbadm";
String empname;              // Declare a host variable
...
try {                        // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                             2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Ctx myConnCtx=                                                            3
  new Ctx("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
  userid,password,false);    // Create connection context object myConnCtx
                             // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                             // Use myConnCtx for executing an SQL statement
```

Figure 30. Using connection technique 1 to connect to a data source

Connection technique 2: This technique uses the JDBC `DriverManager` interface for creating the connection. Use it with any level of the JDBC driver.

1. Execute an SQLJ connection declaration clause.

   This is the same as step 1 on page 1319 in connection technique 1.
2. Load the driver.

   This is the same as step 2 on page 1320 in connection technique 1.
3. Invoke the JDBC `DriverManager.getConnection` method.

   Doing this creates a JDBC connection object for the connection to the data source. You can use any of the forms of `getConnection` that are specified in Connect to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ.

   The meanings of the *url*, *user*, and *password* parameters are the same as the meanings of the parameters in step 3 on page 1320 of connection technique 1.
4. Invoke the constructor for the connection context class that you created in step 1.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in the following form:

   ```
   connection-context-class connection-context-object=
     new connection-context-class(Connection JDBC-connection-object);
   ```

   The *JDBC-connection-object* parameter is the `Connection` object that you created in step 3.

The following code uses connection technique 2 to create a connection to location NEWYORK. The connection requires a user ID and password, and does not require autocommit. The numbers to the right of selected statements correspond to the previously-described steps.

```
#sql context Ctx;           // Create connection context class Ctx       1
String userid="dbadm";      // Declare variables for user ID and password
String password="dbadm";
String empname;             // Declare a host variable
...
try {                       // Load the JDBC driver
  Class.forName("com.ibm.db2.jcc.DB2Driver");                          2
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
Connection jdbccon=                                                     3
  DriverManager.getConnection("jdbc:db2://sysmvs1.stl.ibm.com:5021/NEWYORK",
    userid,password);
                              // Create JDBC connection object jdbccon
jdbccon.setAutoCommit(false); // Do not autocommit                      4
Ctx myConnCtx=new Ctx(jdbccon);                                         5
                              // Create connection context object myConnCtx
                              // for the connection to NEWYORK
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                              // Use myConnCtx for executing an SQL statement
```

Figure 31. Using connection technique 2 to connect to a data source

Connection technique 3: This technique uses the JDBC `DataSource` interface for creating the connection.

1. Execute an SQLJ connection declaration clause.

   This is the same as step 1 on page 1319 in connection technique 1.
2. If your system administrator created a `DataSource` object in a different program:
   a. Obtain the logical name of the data source to which you need to connect.
   b. Create a context to use in the next step.
   c. In your application program, use the Java Naming and Directory Interface (JNDI) to get the `DataSource` object that is associated with the logical data source name.

   Otherwise, create a `DataSource` object and assign properties to it, as shown in ″Creating and using a DataSource object in the same application″ in Connect to a data source using the DataSource interface.
3. Invoke the JDBC `DataSource.getConnection` method.

   Doing this creates a JDBC connection object for the connection to the data source. You can use one of the following forms of `getConnection`:

   ```
   getConnection();
   getConnection(user, password);
   ```

   The meanings of *user* and *password* parameters are the same as the meanings of the parameters in step 3 on page 1320 of connection technique 1.
4. If the default autocommit mode is not appropriate, invoke the JDBC `Connection.setAutoCommit` method.

   Doing this indicates whether you want the database manager to issue a COMMIT after every statement. The form of this method is:

   ```
   setAutoCommit(boolean autocommit);
   ```
5. Invoke the constructor for the connection context class that you created in step 1.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in the following form:

```
connection-context-class connection-context-object=
  new connection-context-class(Connection JDBC-connection-object);
```

The *JDBC-connection-object* parameter is the `Connection` object that you created in step 3 on page 1322.

The following code uses connection technique 3 to create a connection to a location with logical name `jdbc/sampledb`. The numbers to the right of selected statements correspond to the previously-described steps.

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
#sql context CtxSqlj;       // Create connection context class CtxSqlj   1
Context ctx=new InitialContext();                                         2b
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");                    2c
Connection con=ds.getConnection();                                        3
String empname;             // Declare a host variable
...
con.setAutoCommit(false);   // Do not autocommit                          4
CtxSqlj myConnCtx=new CtxSqlj(con);                                       5
                            // Create connection context object myConnCtx
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                            // Use myConnCtx for executing an SQL statement
```

Figure 32. Using connection technique 3 to connect to a data source

Connection technique 4 (IBM DB2 Driver for JDBC and SQLJ only): This technique uses the JDBC `DataSource` interface for creating the connection. This technique **requires** that the `DataSource` is registered with JNDI.

1. From your system administrator, obtain the logical name of the data source to which you need to connect.
2. Execute an SQLJ connection declaration clause.

   For this type of connection, the connection declaration clause needs to be of this form:

   ```
   #sql public static context context-class-name
    with (dataSource="logical-name");
   ```

   The connection context must be declared as public and static. *logical-name* is the data source name that you obtained in step 1.
3. Invoke the constructor for the connection context class that you created in step 2.

   Doing this creates a connection context object that you specify in each SQL statement that you execute at the associated data source. The constructor invocation statement needs to be in one of the following forms:

   ```
   connection-context-class connection-context-object=
     new connection-context-class();

   connection-context-class connection-context-object=
     new connection-context-class (String user,
       String password);
   ```

   The meanings of the *user* and *password* parameters are the same as the meanings of the parameters in step 3 on page 1320 of connection technique 1.

The following code uses connection technique 4 to create a connection to a location with logical name `jdbc/sampledb`. The connection requires a user ID and password.

```
#sql public static context Ctx
  with (dataSource="jdbc/sampledb");                            2
                               // Create connection context class Ctx
String userid="dbadm";         // Declare variables for user ID and password
String password="dbadm";

String empname;                // Declare a host variable
...
Ctx myConnCtx=new Ctx(userid, password);                        3
                               // Create connection context object myConnCtx
                               // for the connection to jdbc/sampledb
#sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};
                               // Use myConnCtx for executing an SQL statement
```

Figure 33. Using connection technique 4 to connect to a data source

Connection technique 5: This technique uses a previously created connection to connect to the data source. In general, one program declares a connection context class, creates connection contexts, and passes them as parameters to other programs. A program that uses the connection context invokes a constructor with the passed connection context object as its argument.

Example: Program CtxGen.sqlj declares connection context Ctx and creates instance oldCtx:

```
#sql context Ctx;
...
// Create connection context object oldCtx
```

Program test.sqlj receives oldCtx as a parameter and uses oldCtx as the argument of its connection context constructor:

```
void useContext(sqlj.runtime.ConnectionContext oldCtx)
                                   // oldCtx was created in CtxGen.sqlj
{
  Ctx myConnCtx=
    new Ctx(oldCtx);               // Create connection context object myConnCtx
                                   // from oldCtx
  #sql [myConnCtx] {SELECT LASTNAME INTO :empname FROM EMPLOYEE
     WHERE EMPNO='000010'};
                                   // Use myConnCtx for executing an SQL statement
...
}
```

Connection technique 6: This technique uses the default connection to connect to the data source. It should be used only in situations where the database thread is controlled by another resource manager, such as the Java stored procedure environment. You use the default connection by specifying your SQL statements without a connection context object. When you use this technique, you do not need to load a JDBC driver unless you explicitly use JDBC interfaces in your program. For example:

```
#sql {SELECT LASTNAME INTO :empname FROM EMPLOYEE
   WHERE EMPNO='000010'};  // Use default connection for
                                      // executing an SQL statement
```

To create a default connection context, SQLJ does a JNDI lookup for `jdbc/defaultDataSource`. If nothing is registered, a null context exception is issued when SQLJ attempts to access the context.

Related concepts:

- "How JDBC applications connect to a data source" on page 1326

Related tasks:

- “Connecting to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ” on page 1342
- “Connecting to a data source using the DataSource interface” on page 1327

Related reference:

- “Properties for the IBM DB2 Driver for JDBC and SQLJ” in *Developing Java Applications*

SQLJ connection-declaration-clause

The connection declaration clause declares a connection to a data source in an SQLJ application program.

Syntax:

```
►►──┬────────────────┬──context──Java-class-name──┬───────────────────┬──┬─────────────┬──────────►◄
    └─Java-modifiers─┘                            └─implements-clause─┘  └─with-clause─┘
```

Description:

Java-modifiers
: Specifies modifiers that are valid for Java class declarations, such as static, public, private, or protected.

Java-class-name
: Specifies a valid Java identifier. During the program preparation process, SQLJ generates a connection context class whose name is this identifier.

implements-clause
: See SQLJ implements-clause for a description of this clause. In a connection declaration clause, the interface class to which the implements clause refers must be a user-defined interface class.

with-clause
: See SQLJ with-clause for a description of this clause.

Usage notes:

- SQLJ generates a connection class declaration for each connection declaration clause you specify. SQLJ data source connections are objects of those generated connection classes.
- You can specify a connection declaration clause anywhere that a Java class definition can appear in a Java program.

Related tasks:

- “Connecting to a data source using SQLJ” on page 1319

Related reference:

- “SQLJ implements-clause” in *Developing Java Applications*
- “SQLJ with-clause” in *Developing Java Applications*

Closing the connection to a data source in an SQLJ application

When you have finished with a connection to a data source, you need to close the connection to the data source. Doing so releases the connection context object’s DB2 and SQLJ resources immediately.

To close the connection to the data source, use one of the `ConnectionContext.close` methods. If you execute `ConnectionContext.close()` or `ConnectionContext.close(ConnectionContext.CLOSE_CONNECTION)`, the connection context, as well as the connection to the data source, are closed. If you execute `ConnectionContext.close(ConnectionContext.KEEP_CONNECTION)` the connection context is closed, but the connection to the data source is not. For example:

```
...
ctx = new EzSqljctx(con0);          // Create a connection context object
                                    // from JDBC connection con0
...                                   // Perform various SQL operations
 EzSqljctx.close(ConnectionContext.KEEP_CONNECTION);
                                    // Close the connection context but keep
                                    // the connection to the data source open
```

Related tasks:

- "Connecting to a data source using SQLJ" on page 1319

JDBC Considerations

How JDBC applications connect to a data source

Before you can execute SQL statements in any SQL program, you must connect to a database server. In JDBC, a database server is known as a *data source*.

Figure 34 shows how a Java application connects to a data source for a type 2 driver or IBM DB2 Driver for JDBC and SQLJ type 2 connectivity.

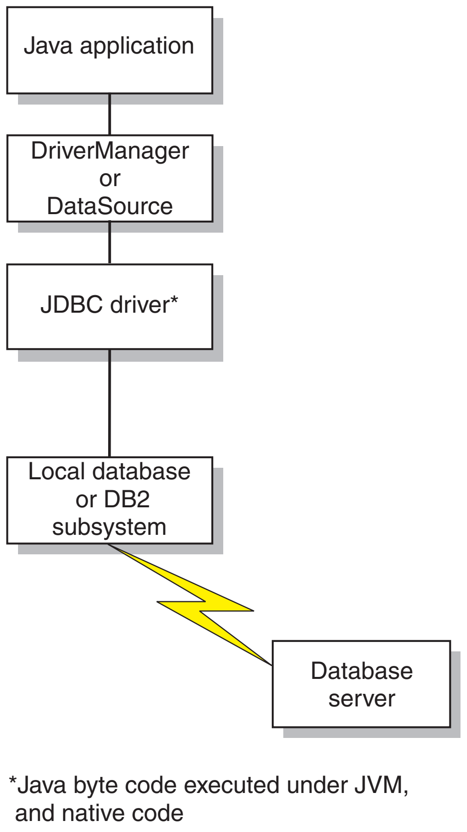

Java application
DriverManager or DataSource
JDBC driver*
Local database or DB2 subsystem
Database server
*Java byte code executed under JVM, and native code

Figure 34. Java application flow for a type 2 driver or IBM DB2 Driver for JDBC and SQLJ type 2 connectivity

Figure 35 shows how a Java application connects to a data source for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity.

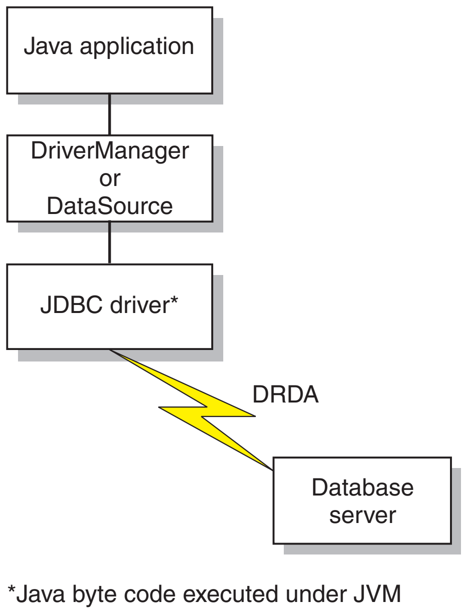

Java application
DriverManager or DataSource
JDBC driver*
DRDA
Database server
*Java byte code executed under JVM

Figure 35. Java application flow for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity

Related concepts:

- "How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver" on page 1332

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 1327
- "Connecting to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ" on page 1342

Connecting to a data source using the DataSource interface

Using `DriverManager` to connect to a data source reduces portability because the application must identify a specific JDBC driver class name and driver URL. The driver class name and driver URL are specific to a JDBC vendor, driver implementation, and data source. If your applications need to be portable among data sources, you should use the `DataSource` interface.

When you connect to a data source using the `DataSource` interface, you use a `DataSource` object. The simplest way to use a `DataSource` object is to create and use the object in the same application, as you do with the `DriverManager` interface. However, this method does not provide portability. Figure 36 shows an example of creating and using a `DataSource` object in the same application.

Figure 36. Creating and using a DataSource object in the same application

```
import java.sql.*;        // JDBC base
import javax.sql.*;       // Methods for producing server-side
                          // applications using Java
import com.ibm.db2.jcc.*; // IBM DB2 Driver for JDBC and SQLJ  1
                          // interfaces
DB2SimpleDataSource db2ds=new DB2SimpleDataSource();      2
```

```
db2ds.setDatabaseName("db2loc1");                          3
                          // Assign the location name
db2ds.setDescription("Our Sample Database");
                          // Description for documentation
db2ds.setUser("john");
                          // Assign the user ID
db2ds.setPassword("db2");
                          // Assign the password
Connection con=db2ds.getConnection();                      4
                          // Create a Connection object
```

Note	Description
1	Import the package that contains the implementation of the DataSource interface.
2	Creates a `DB2SimpleDataSource` object. `DB2SimpleDataSource` is one of the DB2 implementations of the `DataSource` interface. See Create and deploy DataSource objects for information on DB2's `DataSource` implementations.
3	The `setDatabaseName`, `setDescription`, `setUser`, and `setPassword` methods assign attributes to the DB2SimpleDataSource object. See Properties for the IBM DB2 Driver for JDBC and SQLJ for information about the attributes that you can set for a `DB2SimpleDataSource` object under the IBM DB2 Driver for JDBC and SQLJ.
4	Establishes a connection to the data source that `DB2SimpleDataSource` object db2ds represents.

The best way to use a `DataSource` object is for your system administrator to create and manage it separately, using WebSphere® or some other tool. The program that creates and manages a `DataSource` object also uses the Java Naming and Directory Interface (JNDI) to assign a logical name to the `DataSource` object. The JDBC application that uses the `DataSource` object can then refer to the object by its logical name, and does not need any information about the underlying data source. In addition, your system administrator can modify the data source attributes, and you do not need to change your application program.

To learn more about using WebSphere to deploy DataSource objects, go to this URL on the Web:

`http://www.ibm.com/software/webservers/appserv/`

To learn about deploying DataSource objects yourself, see Create and deploy DataSource objects.

You can use the `DataSource` interface and the `DriverManager` interface in the same application, but for maximum portability, it is recommended that you use only the `DataSource` interface to obtain connections.

The remainder of this topic explains how to create a connection using a `DataSource` object, given that the system administrator has already created the object and assigned a logical name to it.

To obtain a connection using a `DataSource` object, you need to follow these steps:

1. From your system administrator, obtain the logical name of the data source to which you need to connect.
2. Create a `Context` object to use in the next step. The `Context` interface is part of the Java Naming and Directory Interface (JNDI), not JDBC.
3. In your application program, use JNDI to get the `DataSource` object that is associated with the logical data source name.
4. Use the `DataSource.getConnection` method to obtain the connection.

   You can use one of the following forms of the `getConnection` method:

   ```
   getConnection();
   getConnection(String user, String password);
   ```

Use the second form if you need to specify a user ID and password for the connection that are different from the ones that were specified when the DataSource was deployed.

Figure 37 shows an example of the code that you need in your application program to obtain a connection using a DataSource object, given that the logical name of the data source that you need to connect to is jdbc/sampledb. The numbers to the right of selected statements correspond to the previously-described steps.

Figure 37. Obtaining a connection using a DataSource object

```
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
...
Context ctx=new InitialContext();                        2
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");   3
Connection con=ds.getConnection();                       4
```

Related tasks:

- "Creating and deploying DataSource objects" in *Developing Java Applications*

Related reference:

- "Properties for the IBM DB2 Driver for JDBC and SQLJ" in *Developing Java Applications*

JDBC connection objects

When you connect to a data source by either connection method, you create a Connection object, which represents the connection to the data source. You use this Connection object to do the following things:

- Create Statement, PreparedStatement, and CallableStatement objects for executing SQL statements. These are discussed in Execute SQL in a JDBC application.
- Gather information about the data source to which you are connected. This process is discussed in Use DatabaseMetaData to learn about a data source.
- Commit or roll back transactions. You can commit transactions manually or automatically. These operations are discussed in Commit or roll back a JDBC transaction.
- Close the connection to the data source. This operation is discussed in Close a connection to a JDBC data source.

Related concepts:

- "JDBC interfaces for executing SQL" in *Developing Java Applications*

Related tasks:

- "Disconnecting from database servers in JDBC applications" on page 1330
- "Committing or rolling back JDBC transactions" on page 1330
- "Learning about a data source using DatabaseMetaData methods" in *Developing Java Applications*

Committing or rolling back JDBC transactions

In JDBC, to commit or roll back transactions explicitly, use the `commit` or `rollback` methods. For example:

```
 Connection con;
 ...
con.commit();
```

If autocommit mode is on, the DB2 database manager performs a commit operation after every SQL statement completes. To set autocommit mode on, invoke the `Connection.setAutoCommit(true)` method. To set autocommit mode off, invoke the `Connection.setAutoCommit(false)` method. To determine whether autocommit mode is on, invoke the `Connection.getAutoCommit` method.

When autocommit mode is on, you cannot execute the `commit` and `rollback` methods.

Connections that participate in distributed transactions cannot invoke the `setAutoCommit(true)` method.

When you change the autocommit state, the DB2 database manager executes a commit operation, if the application is not already on a transaction boundary.

While a connection is participating in a distributed transaction, the associated application cannot issue the `commit` or `rollback` methods.

Related concepts:

- "Savepoints in JDBC applications" in *Developing Java Applications*

Related tasks:

- "Disconnecting from database servers in JDBC applications" on page 1330
- "Making batch updates in JDBC applications" in *Developing Java Applications*

Disconnecting from database servers in JDBC applications

When you have finished with a connection to a data source, it is *essential* that you close the connection to the data source. Doing this releases the `Connection` object's DB2 and JDBC resources immediately. To close the connection to the data source, use the `close` method. For example:

```
 Connection con;
 ...
con.close();
```

If autocommit mode is not on, the connection needs to be on a unit-of-work boundary before you close the connection.

Related concepts:

- "How JDBC applications connect to a data source" on page 1326

Type 2 JDBC Driver Considerations

Security under the DB2 JDBC Type 2 Driver

The DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (DB2 JDBC Type 2 Driver) supports user ID and password security. You must set the user ID and the password, or set neither. If you do not set a user ID and password, the driver uses the user ID and password of the user who is currently logged on to the operating system.

To specify user ID and password security for a JDBC connection, use one of the following techniques.

For the* `DriverManager` *interface: you can specify the user ID and password directly in the `DriverManager.getConnection` invocation. For example:

```
import java.sql.*;        // JDBC base
...
String id = "db2adm";      // Set user ID
Sring pw = "db2adm";       // Set password
String url = "jdbc:db2:toronto";
                           // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                           // Create connection
```

Alternatively, you can set the user ID and password by setting the `user` and `password` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example:

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "db2adm");           // Set user ID for the connection
properties.put("password", "db2adm");       // Set password for the connection
String url = "jdbc:db2:toronto";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

For the* `DataSource` *interface: you can specify the user ID and password directly in the `DataSource.getConnection` invocation. For example:

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC
...
Context ctx=new InitialContext();           // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Get DataSource object
String id = "db2adm";                       // Set user ID
Sring pw = "db2adm";                        // Set password
Connection con = ds.getConnection(id, pw);
                                            // Create connection
```

Alternatively, if you create and deploy the `DataSource` object, you can set the user ID and password by invoking the `DataSource.setUser` and `DataSource.setPassword` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                          // JDBC base
import COM.ibm.db2.jdbc.*;                  // DB2 implementation of JDBC
...
DB2DataSource db2ds = new DB2DataSource();
                                              // Create DataSource object
db2ds.setDatabaseName("toronto");             // Set location
db2ds.setUser("db2adm");                      // Set user ID
db2ds.setPassword("db2adm");                  // Set password
```

Related concepts:

- “How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver” on page 1332

Related tasks:

- “Connecting to a data source using the DataSource interface” on page 1327
- “Creating and deploying DataSource objects” in *Developing Java Applications*

How DB2 applications connect to a data source using the DriverManager interface with the DB2 JDBC Type 2 Driver

A JDBC application can establish a connection to a data source using the JDBC `DriverManager` interface, which is part of the `java.sql` package.

The Java application first loads the JDBC driver by invoking the `Class.forName` method. After the application loads the driver, it connects to a database server by invoking the `DriverManager.getConnection` method.

For the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (DB2 JDBC Type 2 Driver), you load the driver by invoking the `Class.forName` method with the following argument:

```
COM.ibm.db2.jdbc.app.DB2Driver
```

The following code demonstrates loading the DB2 JDBC Type 2 Driver:

```
try {
  // Load the DB2 JDBC Type 2 Driver with DriverManager
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
} catch (ClassNotFoundException e) {
     e.printStackTrace();
}
```

The `catch` block is used to print an error if the driver is not found.

After you load the driver, you connect to the data source by invoking the `DriverManager.getConnection` method. You can use one of the following forms of `getConnection`:

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

The *url* argument represents a data source.

For the DB2 JDBC Type 2 Driver, specify a URL of the following form:

Syntax for a URL for the DB2 JDBC Type 2 Driver:

```
►►──jdbc:db2:database────────────────────────────────────────────────►◄
```

The parts of the URL have the following meanings:

jdbc:db2:
: jdbc:db2: indicates that the connection is to a DB2 database server.

database
: A database alias. The alias refers to the DB2 database catalog entry on the DB2 client.

The *info* argument is an object of type `java.util.Properties` that contains a set of driver properties for the connection. Specifying the *info* argument is an alternative to specifying *property=value* strings in the URL.

Specifying a user ID and password for a connection: There are several ways to specify a user ID and password for a connection:
- Use the form of the `getConnection` method that specifies *user* and *password*.
- Use the form of the `getConnection` method that specifies *info*, after setting the user and password properties in a `java.util.Properties` object.

Example: Setting the user ID and password in user and password parameters:

```
String url = "jdbc:db2:toronto";
                                           // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                           // Create connection
```

Example: Setting the user ID and password in a `java.util.Properties` object:

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2:toronto";
                                           // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                           // Create connection
```

Related concepts:
- "Security under the DB2 JDBC Type 2 Driver" on page 1330

Universal JDBC Driver Considerations

User ID and password security under the IBM DB2 Driver for JDBC and SQLJ

To specify user ID and password security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: You can specify the user ID and password directly in the `DriverManager.getConnection` invocation. For example:

```
import java.sql.*;         // JDBC base
...
String id = "db2adm";      // Set user ID
String pw = "db2adm";      // Set password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                           // Set URL for the data source
Connection con = DriverManager.getConnection(url, id, pw);
                           // Create connection
```

Another method is to set the user ID and password directly in the URL string. For example:

```
import java.sql.*;         // JDBC base
...
String url =
  "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose:user=db2adm;password=db2adm;";
                           // Set URL for the data source
Connection con = DriverManager.getConnection(url);
                           // Create connection
```

Alternatively, you can set the user ID and password by setting the `user` and `password` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. Optionally, you can set the `securityMechanism` property to indicate that you are using user ID and password security. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC
...
Properties properties = new java.util.Properties();
                                            // Create Properties object
properties.put("user", "db2adm");          // Set user ID for the connection
properties.put("password", "db2adm");      // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY +
  ""));
                                            // Set security mechanism to
                                            // user ID and password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                            // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                            // Create connection
```

For the `DataSource` interface: you can specify the user ID and password directly in the `DataSource.getConnection` invocation. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC
...
Context ctx=new InitialContext();          // Create context for JNDI
DataSource ds=(DataSource)ctx.lookup("jdbc/sampledb");
                                            // Get DataSource object
String id = "db2adm";                      // Set user ID
String pw = "db2adm";                       // Set password
Connection con = ds.getConnection(id, pw);
                                            // Create connection
```

Alternatively, if you create and deploy the `DataSource` object, you can set the user ID and password by invoking the `DataSource.setUser` and `DataSource.setPassword` methods after you create the `DataSource` object. Optionally, you can invoke the `DataSource.setSecurityMechanism` method property to indicate that you are using user ID and password security. For example:

```
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =    // Create DB2SimpleDataSource object
  new com.ibm.db2.jcc.DB2SimpleDataSource();
db2ds.setDriverType(4);                        // Set driver type
db2ds.setDatabaseName("san_jose");             // Set location
db2ds.setServerName("mvs1.sj.ibm.com");        // Set server name
db2ds.setPortNumber(5021);                     // Set port number
db2ds.setUser("db2adm");                       // Set user ID
db2ds.setPassword("db2adm");                   // Set password
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY);
                                               // Set security mechanism to
                                               // user ID and password
```

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 1327
- "Connecting to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ" on page 1342
- "Creating and deploying DataSource objects" in *Developing Java Applications*

Related reference:

- “Properties for the IBM DB2 Driver for JDBC and SQLJ” in *Developing Java Applications*

User ID-only security under the IBM DB2 Driver for JDBC and SQLJ

To specify user ID security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: Set the user ID and security mechanism by setting the `user` and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example:

```
import java.sql.*;                          // JDBC base
import com.ibm.db2.jcc.*;                   // DB2 implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");         // Set user ID for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY + ""));
                                          // Set security mechanism to
                                          // user ID only
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, you can set the user ID and security mechanism by invoking the `DataSource.setUser` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create DB2SimpleDataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");   // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setUser("db2adm");                  // Set the user ID
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.USER_ONLY_SECURITY);
                                          // Set security mechanism to
                                          // user ID only
```

Kerberos security under the IBM DB2 Driver for JDBC and SQLJ

JDBC support for Kerberos security is available for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity only.

To enable JDBC support for Kerberos security, you also need to enable the following components of your software development kit (SDK) for Java:
- Java Cryptography Extension
- Java Generic Security Service (JGSS)
- Java Authentication and Authorization Service (JAAS)

See the documentation for your SDK for Java for information on how to enable these components.

There are three ways to specify Kerberos security for a connection:
- With a user ID and password
- Without a user ID or password
- With a delegated credential

Using Kerberos security with a user ID and password:

For this case, Kerberos uses the specified user ID and password to obtain a ticket-granting ticket (TGT) that lets you authenticate to the DB2 server.

You need to set the `user`, `password`, `kerberosServerPrincipal`, and `securityMechanism` properties. The `kerberosServerPrincipal` property specifies the principal name that the DB2 server registers with a Kerberos Key Distribution Center (KDC).

For the ***`DriverManager`*** *interface:* Set the user ID, password, Kerberos server, and security mechanism by setting the `user`, `password`, `kerberosServerPrincipal`, and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the Kerberos security mechanism with a user ID and password:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");         // Set user ID for the connection
properties.put("password", "db2adm");     // Set password for the connection
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Set the Kerberos server
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                          // Set security mechanism to
                                          // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

For the ***`DataSource`*** *interface:* If you create and deploy the `DataSource` object, set the Kerberos server and security mechanism by invoking the `DataSource.setKerberosServerPrincipal` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create the DataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setUser("db2adm");                  // Set the user
db2ds.setPassword("db2adm");              // Set the password
db2ds.setServerName("mvs1.sj.ibm.com");
                                          // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                          // Set security mechanism to
                                          // Kerberos
```

Using Kerberos security with no user ID or password:

For this case, the Kerberos default credentials cache must contain a ticket-granting ticket (TGT) that lets you authenticate to the DB2 server.

You need to set the `kerberosServerPrincipal` and `securityMechanism` properties.

*For the **`DriverManager`** interface:* Set the Kerberos server and security mechanism by setting the `kerberosServerPrincipal` and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the Kerberos security mechanism without a user ID and password:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Set the Kerberos server
properties.put("securityMechanism",
  new String("" +
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                          // Set security mechanism to
                                          // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                          // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create the connection
```

*For the **`DataSource`** interface:* If you create and deploy the `DataSource` object, set the Kerberos server and security mechanism by invoking the `DataSource.setKerberosServerPrincipal` and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
import java.sql.*;                        // JDBC base
import com.ibm.db2.jcc.*;                 // DB2 implementation of JDBC
...
DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                          // Create the DataSource object
db2ds.setDriverType(4);                   // Set the driver type
db2ds.setDatabaseName("san_jose");        // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                          // Set the server name
db2ds.setPortNumber(5021);                // Set the port number
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                          // Set the Kerberos server
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                          // Set security mechanism to
                                          // Kerberos
```

Using Kerberos security with a delegated credential from another principal:

For this case, you authenticate to the DB2 server using a delegated credential that another principal passes to you.

You need to set the `kerberosServerPrincipal`, `gssCredential`, and `securityMechanism` properties.

For the `DriverManager` interface: Set the Kerberos server, delegated credential, and security mechanism by setting the `kerberosServerPrincipal`, and `securityMechanism` properties in a `Properties` object. Then invoke the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the Kerberos security mechanism without a user ID and password:

```
import java.sql.*;                         // JDBC base
import com.ibm.db2.jcc.*;                  // DB2 implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("kerberosServerPrincipal",
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                           // Set the Kerberos server
properties.put("gssCredential",delegatedCredential);
                                           // Set the delegated credential
properties.put("securityMechanism",
  new String("" +
    com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY + ""));
                                           // Set security mechanism to
                                           // Kerberos
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                           // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                           // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, set the Kerberos server, delegated credential, and security mechanism by invoking the `DataSource.setKerberosServerPrincipal`, `DataSource.setGssCredential`, and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example:

```
DB2SimpleDataSource db2ds = new com.ibm.db2.jcc.DB2SimpleDataSource();
                                              // Create the DataSource object
db2ds.setDriverType(4);                       // Set the driver type
db2ds.setDatabaseName("san_jose");            // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");  // Set the server name
db2ds.setPortNumber(5021);                    // Set the port number
db2ds.setKerberosServerPrincipal(
  "sample/srvlsj.ibm.com@SRVLSJ.SJ.IBM.COM");
                                              // Set the Kerberos server
db2ds.setGssCredential(delegatedCredential);
                                              // Set the delegated credential
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.KERBEROS_SECURITY);
                                              // Set security mechanism to
                                              // Kerberos
```

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 1327
- "Connecting to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ" on page 1342
- "Creating and deploying DataSource objects" in *Developing Java Applications*

Related reference:

- "Properties for the IBM DB2 Driver for JDBC and SQLJ" in *Developing Java Applications*

Encrypted password security or encrypted user ID and encrypted password security under the IBM DB2 Driver for JDBC and SQLJ

If you use encrypted password security or encrypted user ID and encrypted password security, the IBM Java Cryptography Extension (ibmjceprovidere.jar) must be installed on your client.

You can also use encrypted security-sensitive data in addition to encrypted user ID security or encrypted password security when you access a DB2 for z/OS server. You specify encryption of security-sensitive data through the `ENCRYPTED_USER_AND_DATA_SECURITY` or `ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY` `securityMechanism` value. DB2 for z/OS encrypts the following data when you specify encryption of security-sensitive data:

- SQL statements that are being prepared, executed, or bound into a DB2 package
- Input and output parameter information
- Result sets
- LOB data
- Results of describe operations

To specify encrypted user ID or encrypted password security for a JDBC connection, use one of the following techniques.

For the `DriverManager` interface: Set the user ID, password, and security mechanism by setting the `user`, `password`, and `securityMechanism` properties in a `Properties` object, and then invoking the form of the `getConnection` method that includes the `Properties` object as a parameter. For example, use code like this to set the user ID and encrypted password security mechanism:

```
import java.sql.*;                       // JDBC base
import com.ibm.db2.jcc.*;                // DB2 implementation of JDBC
...
Properties properties = new Properties(); // Create a Properties object
properties.put("user", "db2adm");        // Set user ID for the connection
properties.put("password", "db2adm");    // Set password for the connection
properties.put("securityMechanism",
  new String("" + com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY +
  ""));
                                         // Set security mechanism to
                                         // user ID and encrypted password
String url = "jdbc:db2://mvs1.sj.ibm.com:5021/san_jose";
                                         // Set URL for the data source
Connection con = DriverManager.getConnection(url, properties);
                                         // Create the connection
```

For the `DataSource` interface: If you create and deploy the `DataSource` object, you can set the user ID, password, and security mechanism by invoking the `DataSource.setUser`, `DataSource.setPassword`, and `DataSource.setSecurityMechanism` methods after you create the `DataSource` object. For example, use code like this to set the encrypted user ID and encrypted password security mechanism:

```
import java.sql.*;                       // JDBC base
import com.ibm.db2.jcc.*;                // DB2 implementation of JDBC
...
com.ibm.db2.jcc.DB2SimpleDataSource db2ds =
  new com.ibm.db2.jcc.DB2SimpleDataSource();
                                         // Create the DataSource object
db2ds.setDriverType(4);                  // Set the driver type
db2ds.setDatabaseName("san_jose");       // Set the location
db2ds.setServerName("mvs1.sj.ibm.com");
                                         // Set the server name
```

```
db2ds.setPortNumber(5021);              // Set the port number
db2ds.setUser("db2adm");                // Set the user ID
db2ds.setPassword("db2adm");            // Set the password
db2ds.setSecurityMechanism(
  com.ibm.db2.jcc.DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY);
                                        // Set security mechanism to
                                        // User ID and encrypted password
```

Related tasks:

- "Connecting to a data source using the DataSource interface" on page 1327
- "Connecting to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ" on page 1342
- "Creating and deploying DataSource objects" in *Developing Java Applications*

Related reference:

- "Properties for the IBM DB2 Driver for JDBC and SQLJ" in *Developing Java Applications*

Security under the IBM DB2 Driver for JDBC and SQLJ

When you use the IBM DB2 Driver for JDBC and SQLJ, you choose a security mechanism by specifying a value for the `securityMechanism` property. You can set this property in one of the following ways:

- If you use the `DriverManager` interface, set `securityMechanism` in a `java.util.Properties` object before you invoke the form of the `getConnection` method that includes the `java.util.Properties` parameter.
- If you use the `DataSource` interface, and you are creating and deploying your own `DataSource` objects, invoke the `DataSource.setSecurityMechanism` method after you create a `DataSource` object.

Table 150 lists the security mechanisms that the IBM DB2 Driver for JDBC and SQLJ supports, and the value that you need to specify for the `securityMechanism` property to specify each security mechanism.

The default security mechanism is CLEAR_TEXT_PASSWORD_SECURITY. If the server does not support CLEAR_TEXT_PASSWORD_SECURITY but supports ENCRYPTED_USER_AND_PASSWORD_SECURITY, the IBM DB2 Driver for JDBC and SQLJ driver updates the security mechanism to ENCRYPTED_USER_AND_PASSWORD_SECURITY and attempts to connect to the server. Any other mismatch in security mechanism support between the requester and the server results in an error.

Table 150. Security mechanisms supported by the IBM DB2 Driver for JDBC and SQLJ

Security mechanism	`securityMechanism` property value
User ID and password	`DB2BaseDataSource.CLEAR_TEXT_PASSWORD_SECURITY`
User ID only	`DB2BaseDataSource.USER_ONLY_SECURITY`
User ID and encrypted password	`DB2BaseDataSource.ENCRYPTED_PASSWORD_SECURITY`
Encrypted user ID and encrypted password	`DB2BaseDataSource.ENCRYPTED_USER_AND_PASSWORD_SECURITY`
Encrypted user ID and encrypted security-sensitive data	`DB2BaseDataSource.ENCRYPTED_USER_AND_DATA_SECURITY`
Encrypted user ID, encrypted password, and encrypted security-sensitive data	`DB2BaseDataSource.ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY`

Table 150. Security mechanisms supported by the IBM DB2 Driver for JDBC and SQLJ (continued)

Security mechanism	`securityMechanism` property value
Kerberos[1]	`DB2BaseDataSource.KERBEROS_SECURITY`
Plugin[2]	`DB2BaseDataSource.PLUGIN_SECURITY`

Note:

1. Available for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity only.
2. Available for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity to DB2 Database for Linux, UNIX, and Windows servers only.

Table 151Tshows possible DB2 Database for Linux, UNIX, and Windows server authentication types and the compatible IBM DB2 Driver for JDBC and SQLJ securityMechanism property values.

Table 151. Compatible DB2 Database for Linux, UNIX, and Windows server authentication types and IBM DB2 Driver for JDBC and SQLJ securityMechanism values

DB2 Database for Linux, UNIX, and Windows server authentication type	securityMechanism setting
CLIENT	USER_ONLY_SECURITY
SERVER	CLEAR_TEXT_PASSWORD_SECURITY
SERVER_ENCRYPT	CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, or ENCRYPTED_USER_AND_PASSWORD_SECURITY
DATA_ENCRYPT	ENCRYPTED_USER_AND_DATA_SECURITY or ENCRYPTED_USER_PASSWORD_AND_DATA_SECURITY
KERBEROS	KERBEROS_SECURITY or PLUGIN_SECURITY[2]
KRB_SERVER_ENCRYPT	KERBEROS_SECURITY , PLUGIN_SECURITY[1], ENCRYPTED_PASSWORD_SECURITY, or ENCRYPTED_USER_AND_PASSWORD_SECURITY
GSSPLUGIN	PLUGIN_SECURITY[1] or KERBEROS_SECURITY
GSS_SERVER_ENCRYPT[3]	CLEAR_TEXT_PASSWORD_SECURITY, ENCRYPTED_PASSWORD_SECURITY, ENCRYPTED_USER_AND_PASSWORD_SECURITY, PLUGIN_SECURITY, or KERBEROS_SECURITY

Notes®:

1. For PLUGIN_SECURITY, the plugin must be a Kerberos plugin.
2. For PLUGIN_SECURITY, one of the plugins at the server identifies itself as supporting Kerberos.
3. GSS_SERVER_ENCRYPT is a combination of GSSPLUGIN and SERVER_ENCRYPT.

Related concepts:

- "Encrypted password security or encrypted user ID and encrypted password security under the IBM DB2 Driver for JDBC and SQLJ" on page 1339
- "Kerberos security under the IBM DB2 Driver for JDBC and SQLJ" on page 1335
- "User ID and password security under the IBM DB2 Driver for JDBC and SQLJ" on page 1333
- "User ID-only security under the IBM DB2 Driver for JDBC and SQLJ" on page 1335

Related reference:

- “Properties for the IBM DB2 Driver for JDBC and SQLJ” in *Developing Java Applications*

Connecting to a data source using the DriverManager interface with the IBM DB2 Driver for JDBC and SQLJ

A JDBC application can establish a connection to a data source using the JDBC `DriverManager` interface, which is part of the `java.sql` package.

The Java application first loads the JDBC driver by invoking the `Class.forName` method. After the application loads the driver, it connects to a database server by invoking the `DriverManager.getConnection` method.

For the IBM DB2 Driver for JDBC and SQLJ, you load the driver by invoking the `Class.forName` method with the following argument:

```
com.ibm.db2.jcc.DB2Driver
```

For compatibility with previous JDBC drivers, you can use the following argument instead:

```
COM.ibm.db2os390.sqlj.jdbc.DB2SQLJDriver
```

The following code demonstrates loading the IBM DB2 Driver for JDBC and SQLJ:

```
try {
  // Load the IBM DB2 Driver for JDBC and SQLJ with DriverManager
  Class.forName("com.ibm.db2.jcc.DB2Driver");
} catch (ClassNotFoundException e) {
     e.printStackTrace();
}
```

The `catch` block is used to print an error if the driver is not found.

After you load the driver, you connect to the data source by invoking the `DriverManager.getConnection` method. You can use one of the following forms of `getConnection`:

```
getConnection(String url);
getConnection(String url, user, password);
getConnection(String url, java.util.Properties info);
```

For IBM DB2 Driver for JDBC and SQLJ type 4 connectivity, the `getConnection` method must specify a user ID and password, through parameters or through property values.

The *url* argument represents a data source, and indicates what type of JDBC connectivity you are using.

For IBM DB2 Driver for JDBC and SQLJ type 4 connectivity, specify a URL of the following form:

Syntax for a URL for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity:

```
►►──┬─jdbc:db2:─────┬─//server──┬───────┬──/database──┬──────────────────────────────┬──────────────►◄
    └─jdbc:db2j:net:┘           └─:port─┘             │   ┌────────────────────────┐ │
                                                      │   ▼                        │ │
                                                      └─:──┴─property──=──value──;─┴─┘
```

For IBM DB2 Driver for JDBC and SQLJ type 2 connectivity, specify a URL of one of the following forms:

Syntax for a URL for IBM DB2 Driver for JDBC and SQLJ type 2 connectivity:

```
►►──┬─┬─jdbc:db2:database───────┬─┬──────────────────────────────────────────►◄
    │ ├─jdbc:db2os390:database──┤ │
    │ ├─jdbc:db2os390sqlj:database┤ │
    │ ├─jdbc:default:connection──┤ │
    │ ├─jdbc:db2os390───────────┤ │
    │ └─jdbc:db2os390sqlj───────┘ │
    │   ┌───────────────────────┐ │
    │   ▼                       │ │
    └─:──property──=──value──;──┴─┘
```

The parts of the URL have the following meanings:

jdbc:db2: or jdbc:db2j:net:
The meanings of the initial portion of the URL are:

jdbc:db2:
Indicates that the connection is to a DB2 for z/OS or DB2 Database for Linux, UNIX, and Windows server.

jdbc:db2j:net:
Indicates that the connection is to a remote IBM Cloudscape™ server.

server
The domain name or IP address of the database server.

port
The TCP/IP server port number that is assigned to the database server. This is an integer between 0 and 65535. The default is 446.

database
A name for the database server. This name depends on whether IBM DB2 Driver for JDBC and SQLJ type 4 connectivity or IBM DB2 Driver for JDBC and SQLJ type 2 connectivity is used.

For IBM DB2 Driver for JDBC and SQLJ type 4 connectivity:

- If the connection is to a DB2 for z/OS server, *database* is the DB2 location name that is defined during installation. All characters in the DB2 location name must be uppercase characters. The IBM DB2 Driver for JDBC and SQLJ does not convert lowercase characters in the database value to uppercase for IBM DB2 Driver for JDBC and SQLJ type 4 connectivity.

  You can determine the location name by executing the following SQL statement on the server:

  ```
  SELECT CURRENT SERVER FROM SYSIBM.SYSDUMMY1;
  ```
- If the connection is to a DB2 for z/OS server, all characters in *database* must be uppercase characters.
- If the connection is to a DB2 Database for Linux, UNIX, and Windows server, *database* is the database name that is defined during installation.
- If the connection is to an IBM Cloudscape server, the *database* is the fully-qualified name of the file that contains the database. This name must be enclosed in double quotation marks ("). For example:

  ```
  "c:/databases/testdb"
  ```

For IBM DB2 Driver for JDBC and SQLJ type 2 connectivity:

- *database* is the database name that is defined during installation, if the value of the serverName connection property is null. If the value of serverName property is not null, *database* is a database alias.
- If the connection is to a DB2 for z/OS server or a DB2 UDB for iSeries server, all characters in *database* must be uppercase characters.

property=value;
A property for the JDBC connection. For the definitions of these properties, see Properties for the IBM DB2 Driver for JDBC and SQLJ.

The *info* argument is an object of type `java.util.Properties` that contains a set of driver properties for the connection. Specifying the *info* argument is an alternative to specifying *property=value* strings in the URL. See Properties for the IBM DB2 Driver for JDBC and SQLJ for the properties that you can specify.

Specifying a user ID and password for a connection: There are several ways to specify a user ID and password for a connection:

- Use the form of the `getConnection` method that specifies *url* with `property=`*value*; clauses, and include the user and password properties in the URL.
- Use the form of the `getConnection` method that specifies *user* and *password*.
- Use the form of the `getConnection` method that specifies *info*, after setting the user and password properties in a `java.util.Properties` object.

Example: Setting the user ID and password in a URL:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose:" +
  "user=db2adm;password=db2adm;";
                                          // Set URL for data source
Connection con = DriverManager.getConnection(url);
                                          // Create connection
```

Example: Setting the user ID and password in user and password parameters:

```
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                                          // Set URL for data source
String user = "db2adm";
String password = "db2adm";
Connection con = DriverManager.getConnection(url, user, password);
                                          // Create connection
```

Example: Setting the user ID and password in a `java.util.Properties` *object:*

```
Properties properties = new Properties(); // Create Properties object
properties.put("user", "db2adm");         // Set user ID for connection
properties.put("password", "db2adm");     // Set password for connection
String url = "jdbc:db2://sysmvs1.stl.ibm.com:5021/san_jose";
                                          // Set URL for data source
Connection con = DriverManager.getConnection(url, properties);
                                          // Create connection
```

Related concepts:

- "Security under the IBM DB2 Driver for JDBC and SQLJ" on page 1340

Related reference:

- "Properties for the IBM DB2 Driver for JDBC and SQLJ" in *Developing Java Applications*

Security and Routines

Benefits of using routines

The following benefits can be gained by using routines:

Encapsulate application logic that can be invoked from an SQL interface
In an environment containing many different client applications that have common requirements, the effective use of routines can simplify code reuse, code standardization, and code maintenance. If a particular aspect of common application behavior needs to be changed in an environment where routines are used, only the affected routine that encapsulates the behavior requires modification. Without routines, application logic changes are required in each application.

Enable controlled access to other database objects
Routines can be used to control access to database objects. A user might not have permission to generally issue a particular SQL statement, such as CREATE TABLE; however the user can be given permission to invoke routines that contain one or more specific implementations of the statement, thus simplifying privilege management through encapsulation of privileges.

Improve application performance by reducing network traffic
When applications run on a client computer, each SQL statement is sent separately from the client computer to the database server computer to be executed and each result set is returned separately. This can result in high levels of network traffic. If a piece of work can be identified that requires extensive database interaction and little user interaction, it makes sense to install this piece of work on the server to minimize the quantity of network traffic and to allow the work to be done on the more powerful database servers.

Allow for faster, more efficient SQL execution
Because routines are database objects, they are more efficient at transmitting SQL requests and data than client applications. Therefore, SQL statements executed within routines can perform better than if executed in client applications. Routines that are created with the NOT FENCED clause run in the same process as the database manager, and can therefore use shared memory for communication, which can result in improved application performance.

Allow the interoperability of logic implemented in different programming languages
Because code modules might be implemented by different programmers in different programming languages, and because it is generally desirable to reuse code when possible, DB2 routines support a high degree of interoperability.

- Client applications in one programming language can invoke routines that are implemented in a different programming language. For example C client applications can invoke .NET common language runtime routines.
- Routines can invoke other routines regardless of the routine type or routine implementation. For example a Java procedure can invoke an embedded SQL scalar function.
- Routines created in a database server on one operating system can be invoked from a DB2 client running on a different operating system.

The benefits described above are just some of the many benefits of using routines. Using routines can be beneficial to a variety of users including database administrators, database architects, and database application developers. For this reason there are many useful applications of routines that you might want to explore.

There are various kinds of routines that address particular functional needs and various routine implementations. The choice of routine type and implementation can impact the degree to which the above benefits are exhibited. In general, routines are a powerful way of encapsulating logic so that you can extend your SQL, and improve the structure, maintenance, and potentially the performance of your applications.

Related concepts:
- "Routine invocation" in *Developing SQL and External Routines*
- "External scalar functions" on page 1348
- "User-defined table functions" on page 1350
- "Supported routine programming languages" in *Developing SQL and External Routines*

Related tasks:
- "Debugging routines" in *Developing SQL and External Routines*
- "Creating routines" in *Developing SQL and External Routines*
- "Writing routines" in *Developing SQL and External Routines*
- "Building routines in C or C++ using the sample build script (UNIX)" in *Developing Embedded SQL Applications*
- "Building C/C++ routines on Windows" in *Developing Embedded SQL Applications*
- "Building IBM COBOL routines on AIX" in *Developing Embedded SQL Applications*
- "Building IBM COBOL routines on Windows" in *Developing Embedded SQL Applications*
- "Building JDBC routines" in *Developing Java Applications*
- "Building UNIX Micro Focus COBOL routines" in *Developing Embedded SQL Applications*
- "Building Micro Focus COBOL routines on Windows" in *Developing Embedded SQL Applications*
- "Building SQLJ routines" in *Developing Java Applications*

Procedures

A procedure, also called a stored procedure, is a database object created by executing the CREATE PROCEDURE statement that encapsulates logic and SQL statements. Procedures are used as subroutine extensions to applications, and other database objects that can contain logic.

Features
- Enables the encapsulation of SQL statements, function invocations, and logic elements that formulate a particular subroutine module that can be reused.
- Procedures can be called from client applications, other routines, triggers and dynamic compound statements. Procedures are called using the CALL statement.
- Procedures can return multiple result sets.

- Procedures can contain SQL statements that read or modify table data in both single and multiple partition databases.
- When a procedure is invoked the SQL and logic within a procedure is executed on the server. Data is only transferred between the client and the database server in the procedure call and in the procedure return. If you have a series of SQL statements to execute within a client application, and the application does not need to do any processing in between the statements, then this series of statements would benefit from being included in a procedure.

  Note: If only one SQL statement is invoked in a procedure, the overhead of setting up this invocation outweighs the benefit in network traffic savings.

Limitations

- Procedures are not intended to be called from within elements of an SQL query. Procedures can only be invoked by using the CALL statement where it is supported. Functions can be used to express logic that transforms column values. Although procedures can return result sets, table functions can be used to return a table within the FROM clause of an SQL query.
- Output arguments of procedure calls cannot be directly used by another SQL statement.
- Procedures cannot preserve state between invocations.

Common uses

- To implement application sub-routines that specifically encapsulate the database logic associated with a particular task. For example, a business application for managing employee information could use a procedure to encapsulate the database operations involved in hiring an employee. Such a procedure could insert employee information into an employee table, a department table, and a benefits table, calculate the employee's weekly pay amount based on an input parameter, and return the weekly pay value as one of the output parameters. Another procedure could contain a statistical analysis of data in the employee table and return result sets that contain the results of the analysis. This use of procedures effectively isolates database tasks from non-database tasks within an application.
- Standardize application logic. If multiple applications must similarly access or modify the database, a procedure can provide a single interface for that access or modification. The procedure is available to be used by all of the applications. If the interface must change to accommodate a change in business logic, only the single procedure must be modified.

Supported languages

- SQL
- C/C++
- Java
- OLE
- COBOL
- .NET common language runtime languages

Note: SQL procedures are supported natively and do not require the installation of a compiler.

Related concepts:
- "Authentication methods for your server" on page 89

Related reference:
- "Restrictions on native XML data store" in *XML Guide*
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

External scalar functions

External scalar functions are scalar functions that have their logic implemented in an external programming language.

These functions can be developed and used to extend the set of existing SQL functions and can be invoked in the same manner as DB2 built-in functions such as LENGTH and COUNT. That is, they can be referenced in SQL statements wherever an expression is valid.

The execution of external scalar function logic takes place on the DB2 database server, however unlike built-in or user-defined SQL scalar functions, the logic of external functions can access the database server filesystem, perform system calls or access a network.

External scalar functions can read SQL data, but cannot modify SQL data.

External scalar functions can be repeatedly invoked for a single reference of the function and can maintain state between these invocations by using a scratchpad, which is a memory buffer. This can be powerful if a function requires some initial, but expensive, setup logic. The setup logic can be done on a first invocation using the scratchpad to store some values that can be accessed or updated in subsequent invocations of the scalar function.

Features of external scalar functions

- Can be referenced as part of an SQL statement anywhere an expression is supported.
- The output of a scalar function can be used directly by the invoking SQL statement.
- For external scalar user-defined functions, state can be maintained between the iterative invocations of the function by using a scratchpad.
- Can provide a performance advantage when used in predicates, because they are executed at the server. If a function can be applied to a candidate row at the server, it can often eliminate the row from consideration before transmitting it to the client machine, reducing the amount of data that must be passed from server to client.

Limitations

- Cannot do transaction management within a scalar function. That is, you cannot issue a COMMIT or a ROLLBACK within a scalar function.
- Cannot return result sets.
- Scalar functions are intended to return a single scalar value per set of inputs.

- External scalar functions are not intended to be used for a single invocation. They are designed such that for a single reference to the function and a given set of inputs, that the function be invoked once per input, and return a single scalar value. On the first invocation, scalar functions can be designed to do some setup work, or store some information that can be accessed in subsequent invocations. SQL scalar functions are better suited to functionality that requires a single invocation.
- In a single partition database external scalar functions can contain SQL statements. These statements can read data from tables, but cannot modify data in tables. If the database has more than one partition then there must be no SQL statements in an external scalar function. SQL scalar functions can contain SQL statements that read or modify data.

Common uses

- Extend the set of DB2 built-in functions.
- Perform logic inside an SQL statement that SQL cannot natively perform.
- Encapsulate a scalar query that is commonly reused as a subquery in SQL statements. For example, given a postal code, search a table for the city where the postal code is found.

Supported languages

- C
- C++
- Java
- OLE
- .NET common language runtime languages

Notes:

1. There is a limited capability for creating aggregate functions. Also known as column functions, these functions receive a set of like values (a column of data) and return a single answer. A user-defined aggregate function can only be created if it is sourced upon a built-in aggregate function. For example, if a distinct type `SHOESIZE` exists that is defined with base type INTEGER, you could define a function, `AVG(SHOESIZE)`, as an aggregate function sourced on the existing built-in aggregate function, `AVG(INTEGER)`.
2. You can also create function that return a row. These are known as row functions and can only be used as a transform function for structured types. The output of a row function is a single row.

Related concepts:

- “Benefits of using routines” on page 1345
- “External function and method features” in *Developing SQL and External Routines*
- “External scalar function and method processing model” in *Developing SQL and External Routines*
- “OLE DB user-defined table functions” in *Developing SQL and External Routines*
- “Scratchpads for external functions and methods” in *Developing SQL and External Routines*
- “External table functions” in *Developing SQL and External Routines*

Related tasks:

- “Invoking scalar functions or methods” in *Developing SQL and External Routines*

- “Invoking user-defined table functions” in *Developing SQL and External Routines*

Related reference:

- “CREATE FUNCTION ” on page 920

User-defined table functions

Like scalar UDFs, a table UDF enables you to extend and customize SQL, but for the purpose of generating a table. Table UDFs can only be invoked in the FROM clause of an SQL statement. Table UDFs accept zero or more typed values as input arguments and return a table.

Table functions are powerful because they enable you to make almost any source of data appear as a DB2 table. A table function can be easily created by writing a program that collects the desired data, filters it according to some input parameters if so desired, and returns it to the DB2 one row at a time.

Features

- Can be referenced as part of an SQL statement FROM clause.
- External table-functions can make operating system calls, read data from files or even access data across a network in a single partitioned database.
- Results can be directly processed by the SQL statement that references the table function.
- SQL table functions can encapsulate SQL statements that modify SQL table data. (Only SQL table functions have this property)
- For a single table function reference, a table function can be invoked multiple times and maintain state between invocations by using a scratchpad.
- Provides a set of data for processing.

Limitations

- Cannot do transaction management. This means that you cannot execute COMMIT or ROLLBACK statements from within a table function.
- Cannot return result sets.
- Not designed for single invocations.
- Can only be used in a FROM clause.
- External table functions can read SQL data, but cannot modify SQL data. SQL table functions can be used to contain statements that modify SQL data.

Common uses

- Encapsulate a complex, but commonly used subquery.
- Provide a tabular interface to non-relational data. For example, read a spreadsheet and produce a table, which could then be inserted into a DB2 table.

Supported languages

- SQL
- C/C++
- Java
- OLE
- OLE DB

- .NET common language runtime languages

Related concepts:
- "Benefits of using routines" on page 1345
- "Scratchpads for external functions and methods" in *Developing SQL and External Routines*
- "External table function processing model" in *Developing SQL and External Routines*

Related reference:
- "CREATE FUNCTION " on page 920

Methods

Methods enable you to define behaviors for structured types. They are like scalar UDFs, but can only be defined for structured types. Methods share all the features of scalar UDFs, in addition to the following features:

Features
- Strongly associated with the structured type.
- Can be sensitive to the dynamic type of the subject type.

Limitations
- Can only return a scalar value.
- Can only be used with structured types.
- Cannot be invoked against typed tables.

Common uses
- Providing operations on structured types.
- Encapsulating the structured type.

Supported languages
- SQL
- C/C++
- Java
- OLE

Security considerations for routines

Developing and deploying routines provides you with an opportunity to greatly improve the performance and effectiveness of your database applications. There can, however, be security risks if the deployment of routines is not managed correctly by the database administrator. The following sections describe security risks and means by which you can mitigate these risks. The security risks are followed by a section on how to safely deploy routines whose security is unknown.

Security risks:

NOT FENCED routines can access database manager resources
NOT FENCED routines run in the same process as the database manager. Because of their close proximity to the database engine, NOT FENCED routines can accidentally or maliciously corrupt the database manager's shared memory, or damage the database control structures. Either form of

damage will cause the database manager to fail. NOT FENCED routines can also corrupt databases and their tables.

To ensure the integrity of the database manager and its databases, you must thoroughly screen routines you intend to register as NOT FENCED. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. The greatest potential for corruption arises when code does not properly manage memory or incorrectly uses static variables. These problems are prevalent in languages other than Java and .NET programming langauges.

In order to register a NOT FENCED routine, the CREATE_NOT_FENCED_ROUTINE authority is required. When granting the CREATE_NOT_FENCED_ROUTINE authority, be aware that the recipient can potentially gain unrestricted access to the database manager and all its resources.

Note: NOT FENCED routines are not supported in Common Criteria compliant configurations.

FENCED THREADSAFE routines can access memory in other FENCED THREADSAFE routines

FENCED THREADSAFE routines run as threads inside a shared process. Each of these routines are able to read the memory used by other routine threads in the same process. Therefore, it is possible for one threaded routine to collect sensitive data from other routines in the threaded process. Another risk inherent in the sharing of a single process, is that one routine thread with flawed memory management can corrupt other routine threads, or cause the entire threaded process to crash.

To ensure the integrity of other FENCED THREADSAFE routines, you must thoroughly screen routines you intend to register as FENCED THREADSAFE. These routines must be fully tested, debugged, and exhibit no unexpected side-effects. In the examination of the routine, pay close attention to memory management and the use of static variables. This is where the greatest potential for corruption lies, particularly in languages other than Java.

In order to register a FENCED THREADSAFE routine, the CREATE_EXTERNAL_ROUTINE authority is required. When granting the CREATE_EXTERNAL_ROUTINE authority, be aware that the recipient can potentially monitor or corrupt the memory of other FENCED THREADSAFE routines.

Write access to the database server by the owner of fenced processes can result in database manager corruption

The user ID under which fenced processes run is defined by the `db2icrt` (create instance) or `db2iupdt` (update instance) system commands. This user ID must not have write access to the directory where routine libraries and classes are stored (in UNIX environments, sqllib/function; in Windows environments, sqllib\function). This user ID must also not have read or write access to any database, operating system, or otherwise critical files and directories on the database server.

If the owner of fenced processes does have write access to various critical resources on the database server, the potential for system corruption exists. For example, a database administrator registers a routine received from an unknown source as FENCED NOT THREADSAFE, thinking that any potential harm can be averted by isolating the routine in its own process. However, the user ID that owns fenced processes has write access to the sqllib/function directory. Users invoke this routine, and unbeknownst to them, it overwrites a library in sqllib/function with an alternate version of a routine body that is registered as NOT FENCED. This second routine has unrestricted access to the entire database manager, and can thereby distribute sensitive information from database tables, corrupt the databases, collect authentication information, or crash the database manager.

Ensure the user ID that owns fenced processes does not have write access to critical files or directories on the database server (especially sqllib/function and the database data directories).

Vulnerability of routine libraries and classes

If access to the directory where routine libraries and classes are stored is not controlled, routine libraries and classes can be deleted or overwritten. As discussed in the previous item, the replacement of a NOT FENCED routine body with a malicious (or poorly coded) routine can severely compromise the stability, integrity, and privacy of the database server and its resources.

To protect the integrity of routines, you must manage access to the directory containing the routine libraries and classes. Ensure that the fewest possible number of users can access this directory and its files. When assigning write access to this directory, be aware that this privilege can provide the owner of the user ID unrestricted access to the database manager and all its resources.

Deploying potentially insecure routines:

If you happen to acquire a routine from an unknown source, be sure you know exactly what it does before you build, register, and invoke it. It is recommend that you register it as FENCED and NOT THREADSAFE unless you have tested it thoroughly, and it exhibits no unexpected side-effects.

If you need to deploy a routine that does not meet the criteria for secure routines, register the routine as FENCED and NOT THREADSAFE. To ensure that database integrity is maintained, FENCED and NOT THREADSAFE routines:

- Run in a separate DB2 process, shared with no other routines. If they abnormally terminate, the database manager will be unaffected.
- Use memory that is distinct from memory used by the database. An inadvertent mistake in a value assignment will not affect the database manager.

Related concepts:

- “External routine library and class management” on page 1354
- “Performance considerations for developing routines” in *Developing SQL and External Routines*
- “Benefits of using routines” on page 1345
- “Restrictions on external routines” in *Developing SQL and External Routines*

Related reference:

- “CREATE FUNCTION ” on page 920
- “CREATE PROCEDURE statement” in *SQL Reference, Volume 2*
- “GRANT (Routine Privileges) ” on page 1089

- “REVOKE (Routine Privileges) ” on page 1129

Connection contexts in SQLJ routines

With the introduction of multithreaded routines in DB2 Universal Database, Version 8, it is important that SQLJ routines avoid the use of the default connection context. That is, each SQL statement must explicitly indicate the ConnectionContext object, and that context must be explicitly instantiated in the Java method. For instance, in previous releases of DB2, a SQLJ routine could be written as follows:

```
class myClass
{
  public static void myRoutine( short myInput )
  {
    DefaultContext ctx = DefaultContext.getDefaultContext();
    #sql { some SQL statement };
  }
}
```

This use of the default context causes all threads in a multithreaded environment to use the same connection context, which, in turn, will result in unexpected failures.

The SQLJ routine above must be changed as follows:

```
#context MyContext;

class myClass
{
  public static void myRoutine( short myInput )
  {
    MyContext ctx = new MyContext( "jdbc:default:connection", false );
    #sql [ctx] { some SQL statement };
    ctx.close();
  }
}
```

This way, each invocation of the routine will create its own unique ConnectionContext (and underlying JDBC connection), which avoids unexpected interference by concurrent threads.

Related concepts:

- “Basic steps in writing an SQLJ application” in *Developing Java Applications*
- “SQL statements in an SQLJ application” in *Developing Java Applications*
- “Java routines” in *Developing SQL and External Routines*

External routine library and class management

To successfully develop and invoke external routines, external routine library and class files must be deployed and managed properly.

External routine library and class file management can be minimal if care is taken when external routines are first created and library and class files deployed.

The main external routine management considerations are the following:

- Deployment of external routine library and class files
- Security of external routine libary and class files
- Resolution of external routine libraries and classes

- Modifications to external routine library and class files
- Backup and restore of external routine library and class files

System administrators, database administrators and database application developers should all take responsibility to ensure that external routine library and class files are secure and correctly preserved during routine development and database administration tasks.

Related concepts:

- "Backup and restore of external routine library and class files" in *Developing SQL and External Routines*
- "Performance considerations for developing routines" in *Developing SQL and External Routines*
- "Deployment of external routine libraries and classes" in *Developing SQL and External Routines*
- "Modifications to external routine library and class files" in *Developing SQL and External Routines*
- "Overview of external routines" in *Developing SQL and External Routines*
- "Resolution of external routine libraries and classes" in *Developing SQL and External Routines*
- "Restrictions on external routines" in *Developing SQL and External Routines*
- "Security considerations for routines" on page 1351
- "Security of external routine library or class files" in *Developing SQL and External Routines*

Related reference:

- "ALTER FUNCTION " on page 848
- "ALTER METHOD " on page 851
- "ALTER PROCEDURE " on page 852
- "CREATE FUNCTION " on page 920
- "CREATE METHOD " on page 937
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*

Rebuilding DB2 routine shared libraries

DB2 will cache the shared libraries used for stored procedures and user-defined functions once loaded. If you are developing a routine, you might want to test loading the same shared library a number of times, and this caching can prevent you from picking up the latest version of a shared library. The way to avoid caching problems depends on the type of routine:

1. **Fenced, not threadsafe routines.** The database manager configuration keyword KEEPFENCED has a default value of YES. This keeps the fenced mode process alive. This default setting can interfere with reloading the library. It is best to change the value of this keyword to NO while developing fenced, not threadsafe routines, and then change it back to YES when you are ready to load the final version of your shared library. For more information, see "Updating the database manager configuration file" on page 1356.
2. **Trusted or threadsafe routines.** Except for SQL routines (including SQL procedures), the only way to ensure that an updated version of a DB2 routine library is picked up when that library is used for trusted, or threadsafe routines, is to recycle the DB2 instance by entering db2stop followed by

db2start on the command line. This is not needed for an SQL routine because when it is recreated, the compiler uses a new unique library name to prevent possible conflicts.

For routines other than SQL routines, you can also avoid caching problems by creating the new version of the routine with a differently named library (for example foo.a becomes foo.1.a), and then using either the ALTER PROCEDURE or ALTER FUNCTION SQL statement with the new library.

Related tasks:
- "Updating the database manager configuration file" on page 1356

Related reference:
- "ALTER FUNCTION " on page 848
- "ALTER PROCEDURE " on page 852

Updating the database manager configuration file

This file contains important settings for application development.

The keyword `KEEPFENCED` has the default value `YES`. For fenced, not threadsafe routines (stored procedures and UDFs), this keeps the routine process alive. It is best to change the value of this keyword to `NO` while developing these routines, and then change it back to `YES` when you are ready to load the final version of your shared library. For more information, see "Rebuilding DB2 routine shared libraries" on page 1355.

Note: `KEEPFENCED` was known as `KEEPDARI` in previous versions of DB2.

For Java application development, you need to update the `JDK_PATH` keyword with the path where the Java Development Kit is installed.

Note: `JDK_PATH` was known as `JDK11_PATH` in previous versions of DB2.

Procedure:

To change these settings enter:

```
db2 update dbm cfg using <keyword> <value>
```

For example, to set the keyword `KEEPFENCED` to `NO`:

```
db2 update dbm cfg using KEEPFENCED NO
```

To set the `JDK_PATH` keyword to the directory `/home/db2inst/jdk13`:

```
db2 update dbm cfg using JDK_PATH /home/db2inst/jdk13
```

To view the current settings in the database manager configuration file, enter:

```
db2 get dbm cfg
```

Note: On Windows, you need to enter these commands in a DB2 command window.

Related concepts:
- "Rebuilding DB2 routine shared libraries" on page 1355

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "CREATE FUNCTION " on page 920
- "CREATE PROCEDURE statement" in *SQL Reference, Volume 2*

SQLCA (SQL communications area)

An SQLCA is a collection of variables that is updated at the end of the execution of every SQL statement. A program that contains executable SQL statements and is precompiled with option LANGLEVEL SAA1 (the default) or MIA must provide exactly one SQLCA, though more than one SQLCA is possible by having one SQLCA per thread in a multi-threaded application.

When a program is precompiled with option LANGLEVEL SQL92E, an SQLCODE or SQLSTATE variable may be declared in the SQL declare section or an SQLCODE variable can be declared somewhere in the program.

An SQLCA should not be provided when using LANGLEVEL SQL92E. The SQL INCLUDE statement can be used to provide the declaration of the SQLCA in all languages but REXX. The SQLCA is automatically provided in REXX.

To display the SQLCA after each command executed through the command line processor, issue the command db2 -a. The SQLCA is then provided as part of the output for subsequent commands. The SQLCA is also dumped in the db2diag.log file.

SQLCA field descriptions

Table 152. Fields of the SQLCA. The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlcaid	CHAR(8)	An "eye catcher" for storage dumps containing 'SQLCA'. The sixth byte is 'L' if line number information is returned from parsing an SQL procedure body.
sqlcabc	INTEGER	Contains the length of the SQLCA, 136.
sqlcode	INTEGER	Contains the SQL return code. **Code** **Means** **0** Successful execution (although one or more SQLWARN indicators may be set). **positive** Successful execution, but with a warning condition. **negative** Error condition.
sqlerrml	SMALLINT	Length indicator for *sqlerrmc*, in the range 0 through 70. 0 means that the value of *sqlerrmc* is not relevant.

Table 152. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlerrmc	VARCHAR (70)	Contains one or more tokens, separated by X'FF', which are substituted for variables in the descriptions of error conditions. This field is also used when a successful connection is completed. When a NOT ATOMIC compound SQL statement is issued, it may contain information on up to seven errors. The last token might be followed by X'FF'. The *sqlerrml* value will include any trailing X'FF'.
sqlerrp	CHAR(8)	Begins with a three-letter identifier indicating the product, followed by five digits indicating the version, release, and modification level of the product. For example, SQL09010 means DB2 V9.1 (version 9, release 1, modification level 0). If SQLCODE indicates an error condition, this field identifies the module that returned the error. This field is also used when a successful connection is completed.
sqlerrd	ARRAY	Six INTEGER variables that provide diagnostic information. These values are generally empty if there are no errors, except for sqlerrd(6) from a partitioned database.
sqlerrd(1)	INTEGER	If connection is invoked and successful, contains the maximum expected difference in length of mixed character data (CHAR data types) when converted to the database code page from the application code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. On successful return from an SQL procedure, contains the return status value from the SQL procedure.
sqlerrd(2)	INTEGER	If connection is invoked and successful, contains the maximum expected difference in length of mixed character data (CHAR data types) when converted to the application code page from the database code page. A value of 0 or 1 indicates no expansion; a value greater than 1 indicates a possible expansion in length; a negative value indicates a possible contraction. If the SQLCA results from a NOT ATOMIC compound SQL statement that encountered one or more errors, the value is set to the number of statements that failed.

Table 152. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlerrd(3)	INTEGER	If PREPARE is invoked and successful, contains an estimate of the number of rows that will be returned. After INSERT, UPDATE, DELETE, or MERGE, contains the actual number of rows that qualified for the operation. If compound SQL is invoked, contains an accumulation of all sub-statement rows. If CONNECT is invoked, contains 1 if the database can be updated, or 2 if the database is read only. If the OPEN statement is invoked, and the cursor contains SQL data change statements, this field contains the sum of the number of rows that qualified for the embedded insert, update, delete, or merge operations. If CREATE PROCEDURE for an SQL procedure is invoked, and an error is encountered when parsing the SQL procedure body, contains the line number where the error was encountered. The sixth byte of sqlcaid must be 'L' for this to be a valid line number.
sqlerrd(4)	INTEGER	If PREPARE is invoked and successful , contains a relative cost estimate of the resources required to process the statement. If compound SQL is invoked, contains a count of the number of successful sub-statements. If CONNECT is invoked, contains 0 for a one-phase commit from a down-level client; 1 for a one-phase commit; 2 for a one-phase, read-only commit; and 3 for a two-phase commit.
sqlerrd(5)	INTEGER	Contains the total number of rows deleted, inserted, or updated as a result of both: • The enforcement of constraints after a successful delete operation • The processing of triggered SQL statements from activated triggers If compound SQL is invoked, contains an accumulation of the number of such rows for all sub-statements. In some cases, when an error is encountered, this field contains a negative value that is an internal error pointer. If CONNECT is invoked, contains an authentication type value of 0 for server authentication; 1 for client authentication; 2 for authentication using DB2 Connect; 4 for SERVER_ENCRYPT authentication; 5 for authentication using DB2 Connect with encryption; 7 for KERBEROS authentication; 9 for GSSPLUGIN authentication; 11 for DATA_ENCRYPT authentication; and 255 for unspecified authentication.
sqlerrd(6)	INTEGER	For a partitioned database, contains the partition number of the database partition that encountered the error or warning. If no errors or warnings were encountered, this field contains the partition number of the coordinator partition. The number in this field is the same as that specified for the database partition in the db2nodes.cfg file.
sqlwarn	Array	A set of warning indicators, each containing a blank or W. If compound SQL is invoked, contains an accumulation of the warning indicators set for all sub-statements.

Table 152. Fields of the SQLCA (continued). The field names shown are those present in an SQLCA that is obtained via an INCLUDE statement.

Name	Data Type	Field Values
sqlwarn0	CHAR(1)	Blank if all other indicators are blank; contains 'W' if at least one other indicator is not blank.
sqlwarn1	CHAR(1)	Contains 'W' if the value of a string column was truncated when assigned to a host variable. Contains 'N' if the null terminator was truncated. Contains 'A' if the CONNECT or ATTACH is successful, and the authorization name for the connection is longer than 8 bytes. Contains 'P' if the PREPARE statement relative cost estimate stored in sqlerrd(4) exceeded the value that could be stored in an INTEGER or was less than 1, and either the CURRENT EXPLAIN MODE or the CURRENT EXPLAIN SNAPSHOT special register is set to a value other than NO.
sqlwarn2	CHAR(1)	Contains 'W' if null values were eliminated from the argument of a column function. [a] If CONNECT is invoked and successful, contains 'D' if the database is in quiesce state, or 'I' if the instance is in quiesce state.
sqlwarn3	CHAR(1)	Contains 'W' if the number of columns is not equal to the number of host variables. Contains 'Z' if the number of result set locators specified on the ASSOCIATE LOCATORS statement is less than the number of result sets returned by a procedure.
sqlwarn4	CHAR(1)	Contains 'W' if a prepared UPDATE or DELETE statement does not include a WHERE clause.
sqlwarn5	CHAR(1)	Contains 'E' if an error was tolerated during SQL statement execution.
sqlwarn6	CHAR(1)	Contains 'W' if the result of a date calculation was adjusted to avoid an impossible date.
sqlwarn7	CHAR(1)	Reserved for future use. If CONNECT is invoked and successful, contains 'E' if the DYN_QUERY_MGMT database configuration parameter is enabled.
sqlwarn8	CHAR(1)	Contains 'W' if a character that could not be converted was replaced with a substitution character.
sqlwarn9	CHAR(1)	Contains 'W' if arithmetic expressions with errors were ignored during column function processing.
sqlwarn10	CHAR(1)	Contains 'W' if there was a conversion error when converting a character data value in one of the fields in the SQLCA.
sqlstate	CHAR(5)	A return code that indicates the outcome of the most recently executed SQL statement.

[a] Some functions may not set SQLWARN2 to W, even though null values were eliminated, because the result was not dependent on the elimination of null values.

Error reporting

The order of error reporting is as follows:

1. Severe error conditions are always reported. When a severe error is reported, there are no additions to the SQLCA.

2. If no severe error occurs, a deadlock error takes precedence over other errors.
3. For all other errors, the SQLCA for the first negative SQL code is returned.
4. If no negative SQL codes are detected, the SQLCA for the first warning (that is, positive SQL code) is returned.

   In a partitioned database system, the exception to this rule occurs if a data manipulation operation is invoked against a table that is empty on one database partition, but has data on other database partitions. SQLCODE +100 is only returned to the application if agents from all database partitions return SQL0100W, either because the table is empty on all database partitions, or there are no more rows that satisfy the WHERE clause in an UPDATE statement.

SQLCA usage in partitioned database systems

In partitioned database systems, one SQL statement may be executed by a number of agents on different database partitions, and each agent may return a different SQLCA for different errors or warnings. The coordinator agent also has its own SQLCA.

To provide a consistent view for applications, all SQLCA values are merged into one structure, and SQLCA fields indicate global counts, such that:

- For all errors and warnings, the *sqlwarn* field contains the warning flags received from all agents.
- Values in the *sqlerrd* fields indicating row counts are accumulations from all agents.

Note that SQLSTATE 09000 may not be returned every time an error occurs during the processing of a triggered SQL statement.

SQLDA (SQL descriptor area)

An SQLDA is a collection of variables that is required for execution of the SQL DESCRIBE statement. The SQLDA variables are options that can be used by the PREPARE, OPEN, FETCH, and EXECUTE statements. An SQLDA communicates with dynamic SQL; it can be used in a DESCRIBE statement, modified with the addresses of host variables, and then reused in a FETCH or EXECUTE statement.

SQLDAs are supported for all languages, but predefined declarations are provided only for C, REXX, FORTRAN, and COBOL.

The meaning of the information in an SQLDA depends on its use. In PREPARE and DESCRIBE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, and FETCH, an SQLDA describes host variables.

In DESCRIBE and PREPARE, if any one of the columns being described is either a LOB type (LOB locators and file reference variables do not require doubled SQLDAs), reference type, or a user-defined type, the number of SQLVAR entries for the entire SQLDA will be doubled. For example:

- When describing a table with 3 VARCHAR columns and 1 INTEGER column, there will be 4 SQLVAR entries
- When describing a table with 2 VARCHAR columns, 1 CLOB column, and 1 integer column, there will be 8 SQLVAR entries

In EXECUTE, FETCH, and OPEN, if any one of the variables being described is a LOB type (LOB locators and file reference variables do not require doubled SQLDAs) or a structured type, the number of SQLVAR entries for the entire SQLDA must be doubled. (Distinct types and reference types are not relevant in these cases, because the additional information in the double entries is not required by the database.)

SQLDA field descriptions

An SQLDA consists of four variables followed by an arbitrary number of occurrences of a sequence of variables collectively named SQLVAR. In OPEN, FETCH, and EXECUTE, each occurrence of SQLVAR describes a host variable. In DESCRIBE and PREPARE, each occurrence of SQLVAR describes a column of a result table or a parameter marker. There are two types of SQLVAR entries:

- **Base SQLVARs:** These entries are always present. They contain the base information about the column, parameter marker, or host variable such as data type code, length attribute, column name, host variable address, and indicator variable address.
- **Secondary SQLVARs:** These entries are only present if the number of SQLVAR entries is doubled as per the rules outlined above. For user-defined types (distinct or structured), they contain the user-defined type name. For reference types, they contain the target type of the reference. For LOBs, they contain the length attribute of the host variable and a pointer to the buffer that contains the actual length. (The distinct type and LOB information does not overlap, so distinct types can be based on LOBs without forcing the number of SQLVAR entries on a DESCRIBE to be tripled.) If locators or file reference variables are used to represent LOBs, these entries are not necessary.

In SQLDAs that contain both types of entries, the base SQLVARs are in a block before the block of secondary SQLVARs. In each, the number of entries is equal to the value in SQLD (even though many of the secondary SQLVAR entries may be unused).

The circumstances under which the SQLVAR entries are set by DESCRIBE is detailed in “Effect of DESCRIBE on the SQLDA” on page 1366.

Fields in the SQLDA header

Table 153. Fields in the SQLDA Header

C Name	SQL Data Type	Usage in DESCRIBE and PREPARE (set by the database manager except for SQLN)	Usage in FETCH, OPEN, and EXECUTE (set by the application prior to executing the statement)
sqldaid	CHAR(8)	The seventh byte of this field is a flag byte named SQLDOUBLED. The database manager sets SQLDOUBLED to the character ’2’ if two SQLVAR entries have been created for each column; otherwise it is set to a blank (X'20' in ASCII, X'40' in EBCDIC). See “Effect of DESCRIBE on the SQLDA” on page 1366 for details on when SQLDOUBLED is set.	The seventh byte of this field is used when the number of SQLVARs is doubled. It is named SQLDOUBLED. If any of the host variables being described is a structured type, BLOB, CLOB, or DBCLOB, the seventh byte must be set to the character ’2’; otherwise it can be set to any character but the use of a blank is recommended.
sqldabc	INTEGER	For 32 bit, the length of the SQLDA, equal to SQLN*44+16. For 64 bit, the length of the SQLDA, equal to SQLN*56+16	For 32 bit, the length of the SQLDA, >= to SQLN*44+16. For 64 bit, the length of the SQLDA, >= to SQLN*56+16.

Table 153. Fields in the SQLDA Header (continued)

C Name	SQL Data Type	Usage in DESCRIBE and PREPARE (set by the database manager except for SQLN)	Usage in FETCH, OPEN, and EXECUTE (set by the application prior to executing the statement)
sqln	SMALLINT	Unchanged by the database manager. Must be set to a value greater than or equal to zero before the DESCRIBE statement is executed. Indicates the total number of occurrences of SQLVAR.	Total number of occurrences of SQLVAR provided in the SQLDA. SQLN must be set to a value greater than or equal to zero.
sqld	SMALLINT	Set by the database manager to the number of columns in the result table or to the number of parameter markers.	The number of host variables described by occurrences of SQLVAR.

Fields in an occurrence of a base SQLVAR

Table 154. Fields in a Base SQLVAR

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
sqltype	SMALLINT	Indicates the data type of the column or parameter marker, and whether it can contain nulls. (Parameter markers are always considered nullable.) Table 156 on page 1367 lists the allowable values and their meanings. Note that for a distinct or reference type, the data type of the base type is placed into this field. For a structured type, the data type of the result of the FROM SQL transform function of the transform group (based on the CURRENT DEFAULT TRANSFORM GROUP special register) for the type is placed into this field. There is no indication in the base SQLVAR that it is part of the description of a user-defined type or reference type.	Same for host variable. Host variables for datetime values must be character string variables. For FETCH, a datetime type code means a fixed-length character string. If sqltype is an even number value, the sqlind field is ignored.
sqllen	SMALLINT	The length attribute of the column or parameter marker. For datetime columns and parameter markers, the length of the string representation of the values. See Table 156 on page 1367. Note that the value is set to 0 for large object strings (even for those whose length attribute is small enough to fit into a two byte integer).	The length attribute of the host variable. See Table 156 on page 1367. Note that the value is ignored by the database manager for CLOB, DBCLOB, and BLOB columns. The len.sqllonglen field in the Secondary SQLVAR is used instead.

Table 154. Fields in a Base SQLVAR (continued)

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
sqldata	pointer	For string SQLVARS, sqldata contains the code page. For character-string SQLVARs where the column is defined with the FOR BIT DATA attribute, sqldata contains 0. For other character-string SQLVARS, sqldata contains either the SBCS code page for SBCS data, or the SBCS code page associated with the composite MBCS code page for MBCS data. For Japanese EUC, Traditional Chinese EUC, and Unicode UTF-8 character-string SQLVARS, sqldata contains 954, 964, and 1208 respectively. For all other column types, sqldata is undefined.	Contains the address of the host variable (where the fetched data will be stored).
sqlind	pointer	For character-string SQLVARS, sqlind contains 0, except for MBCS data, when sqlind contains the DBCS code page associated with the composite MBCS code page. For all other types, sqlind is undefined.	Contains the address of an associated indicator variable, if there is one; otherwise, not used. If sqltype is an even number value, the sqlind field is ignored.
sqlname	VARCHAR (30)	Contains the unqualified name of the column or parameter marker. For columns and parameter markers that have a system-generated name, the thirtieth byte is set to X'FF'. For column names specified by the AS clause, this byte is X'00'.	When connecting to a host database, sqlname can be set to indicate a FOR BIT DATA string as follows: • The sixth byte of the SQLDAID in the SQLDA header is set to '+' • The length of sqlname is 8 • The first two bytes of sqlname are X'0000' • The third and fourth bytes of sqlname are X'0000' • The remaining four bytes of sqlname are reserved and should be set to X'00000000' When working with XML data, sqlname can be set to indicate an XML subtype as follows: • The length of sqlname is 8 • The first two bytes of sqlname are X'0000' • The third and fourth bytes of sqlname are X'0000' • The fifth byte of sqlname is X'01' • The remaining three bytes of sqlname are reserved and should be set to X'000000'

Fields in an occurrence of a secondary SQLVAR

Table 155. Fields in a Secondary SQLVAR

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
len.sqllonglen	INTEGER	The length attribute of a BLOB, CLOB, or DBCLOB column or parameter marker.	The length attribute of a BLOB, CLOB, or DBCLOB host variable. The database manager ignores the SQLLEN field in the Base SQLVAR for the data types. The length attribute stores the number of bytes for a BLOB or CLOB, and the number of double-byte characters for a DBCLOB.
reserve2	CHAR(3) for 32 bit, and CHAR(11) for 64 bit.	Not used.	Not used.
sqlflag4	CHAR(1)	The value is X'01' if the SQLVAR represents a reference type with a target type named in sqldatatype_name. The value is X'12' if the SQLVAR represents a structured type, with the user-defined type name in sqldatatype_name. Otherwise, the value is X'00'.	Set to X'01' if the SQLVAR represents a reference type with a target type named in sqldatatype_name. Set to X'12' if the SQLVAR represents a structured type, with the user-defined type name in sqldatatype_name. Otherwise, the value is X'00'.
sqldatalen	pointer	Not used.	Used for BLOB, CLOB, and DBCLOB host variables only. If this field is NULL, then the actual length (in double-byte characters) should be stored in the 4 bytes immediately before the start of the data and SQLDATA should point to the first byte of the field length. If this field is not NULL, it contains a pointer to a 4 byte long buffer that contains the actual length *in bytes* (even for DBCLOB) of the data in the buffer pointed to from the SQLDATA field in the matching base SQLVAR. Note that, whether or not this field is used, the len.sqllonglen field must be set.
sqldatatype_name	VARCHAR(27)	For a user-defined type, the database manager sets this to the fully qualified user-defined type name.[1] For a reference type, the database manager sets this to the fully qualified type name of the target type of the reference.	For structured types, set to the fully qualified user-defined type name in the format indicated in the table note.[1]

Table 155. Fields in a Secondary SQLVAR (continued)

Name	Data Type	Usage in DESCRIBE and PREPARE	Usage in FETCH, OPEN, and EXECUTE
reserved	CHAR(3)	Not used.	Not used.

[1] The first 8 bytes contain the schema name of the type (extended to the right with spaces, if necessary). Byte 9 contains a dot (.). Bytes 10 to 27 contain the low order portion of the type name, which is *not* extended to the right with spaces.

Note that, although the prime purpose of this field is for the name of user-defined types, the field is also set for IBM predefined data types. In this case, the schema name is SYSIBM, and the low order portion of the name is the name stored in the TYPENAME column of the DATATYPES catalog view. For example:

```
type name        length   sqldatatype_name
---------        ------   ----------------
A.B              10       A       .B
INTEGER          16       SYSIBM  .INTEGER
"Frank's".SMINT  13       Frank's .SMINT
MY."type  "      15       MY      .type
```

Effect of DESCRIBE on the SQLDA

For a DESCRIBE OUTPUT or PREPARE OUTPUT INTO statement, the database manager always sets SQLD to the number of columns in the result set, or the number of output parameter markers. For a DESCRIBE INPUT or PREPARE INPUT INTO statement, the database manager always sets SQLD to the number of input parameter markers in the statement. Note that a parameter marker that corresponds to an INOUT parameter in a CALL statement is described in both the input and output descriptors.

The SQLVARs in the SQLDA are set in the following cases:

- SQLN >= SQLD and no entry is either a LOB, user-defined type or reference type

  The first SQLD SQLVAR entries are set and SQLDOUBLED is set to blank.
- SQLN >= 2*SQLD and at least one entry is a LOB, user-defined type or reference type

  Two times SQLD SQLVAR entries are set, and SQLDOUBLED is set to ’2’.
- SQLD <= SQLN < 2*SQLD and at least one entry is a distinct type or reference type, but there are no LOB entries or structured type entries

  The first SQLD SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +237 (SQLSTATE 01594) is issued.

The SQLVARs in the SQLDA are NOT set (requiring allocation of additional space and another DESCRIBE) in the following cases:

- SQLN < SQLD and no entry is either a LOB, user-defined type or reference type

  No SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +236 (SQLSTATE 01005) is issued.

  Allocate SQLD SQLVARs for a successful DESCRIBE.
- SQLN < SQLD and at least one entry is a distinct type or reference type, but there are no LOB entries or structured type entries

  No SQLVAR entries are set and SQLDOUBLED is set to blank. If the SQLWARN bind option is YES, a warning SQLCODE +239 (SQLSTATE 01005) is issued.

  Allocate 2*SQLD SQLVARs for a successful DESCRIBE including the names of the distinct types and target types of reference types.

- SQLN < 2*SQLD and at least one entry is a LOB or a structured type

  No SQLVAR entries are set and SQLDOUBLED is set to blank. A warning SQLCODE +238 (SQLSTATE 01005) is issued (regardless of the setting of the SQLWARN bind option).

  Allocate 2*SQLD SQLVARs for a successful DESCRIBE.

References in the above lists to LOB entries include distinct type entries whose source type is a LOB type.

The SQLWARN option of the BIND or PREP command is used to control whether the DESCRIBE (or PREPARE INTO) will return the warning SQLCODEs +236, +237, +239. It is recommended that your application code always consider that these SQLCODEs could be returned. The warning SQLCODE +238 is always returned when there are LOB or structured type entries in the select list and there are insufficient SQLVARs in the SQLDA. This is the only way the application can know that the number of SQLVARs must be doubled because of a LOB or structured type entry in the result set.

If a structured type entry is being described, but no FROM SQL transform is defined (either because no TRANSFORM GROUP was specified using the CURRENT DEFAULT TRANSFORM GROUP special register (SQLSTATE 42741), or because the name group does not have a FROM SQL transform function defined (SQLSTATE 42744), the DESCRIBE will return an error. This error is the same error returned for a DESCRIBE of a table with a structured type entry.

SQLTYPE and SQLLEN

Table 156 shows the values that may appear in the SQLTYPE and SQLLEN fields of the SQLDA. In DESCRIBE and PREPARE INTO, an even value of SQLTYPE means that the column does not allow nulls, and an odd value means the column does allow nulls. In FETCH, OPEN, and EXECUTE, an even value of SQLTYPE means that no indicator variable is provided, and an odd value means that SQLIND contains the address of an indicator variable.

Table 156. SQLTYPE and SQLLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE

	For DESCRIBE and PREPARE INTO		**For FETCH, OPEN, and EXECUTE**	
SQLTYPE	**Column Data Type**	**SQLLEN**	**Host Variable Data Type**	**SQLLEN**
384/385	date	10	fixed-length character string representation of a date	length attribute of the host variable
388/389	time	8	fixed-length character string representation of a time	length attribute of the host variable
392/393	timestamp	26	fixed-length character string representation of a timestamp	length attribute of the host variable
396/397	DATALINK	length attribute of the column	DATALINK	length attribute of the host variable
400/401	N/A	N/A	NUL-terminated graphic string	length attribute of the host variable
404/405	BLOB	0 *	BLOB	Not used. *
408/409	CLOB	0 *	CLOB	Not used. *

Table 156. SQLTYPE and SQLLEN values for DESCRIBE, FETCH, OPEN, and EXECUTE (continued)

	For DESCRIBE and PREPARE INTO		For FETCH, OPEN, and EXECUTE	
SQLTYPE	Column Data Type	SQLLEN	Host Variable Data Type	SQLLEN
412/413	DBCLOB	0 *	DBCLOB	Not used. *
448/449	varying-length character string	length attribute of the column	varying-length character string	length attribute of the host variable
452/453	fixed-length character string	length attribute of the column	fixed-length character string	length attribute of the host variable
456/457	long varying-length character string	length attribute of the column	long varying-length character string	length attribute of the host variable
460/461	N/A	N/A	NUL-terminated character string	length attribute of the host variable
464/465	varying-length graphic string	length attribute of the column	varying-length graphic string	length attribute of the host variable
468/469	fixed-length graphic string	length attribute of the column	fixed-length graphic string	length attribute of the host variable
472/473	long varying-length graphic string	length attribute of the column	long graphic string	length attribute of the host variable
480/481	floating point	8 for double precision, 4 for single precision	floating point	8 for double precision, 4 for single precision
484/485	packed decimal	precision in byte 1; scale in byte 2	packed decimal	precision in byte 1; scale in byte 2
492/493	big integer	8	big integer	8
496/497	large integer	4	large integer	4
500/501	small integer	2	small integer	2
916/917	Not applicable	Not applicable	BLOB file reference variable.	267
920/921	Not applicable	Not applicable	CLOB file reference variable.	267
924/925	Not applicable	Not applicable	DBCLOB file reference variable.	267
960/961	Not applicable	Not applicable	BLOB locator	4
964/965	Not applicable	Not applicable	CLOB locator	4
968/969	Not applicable	Not applicable	DBCLOB locator	4
988/989	XML	0	None applicable. Use an XML AS <string or binary LOB type> host variable instead.	Not used.

Note:

- The len.sqllonglen field in the secondary SQLVAR contains the length attribute of the column.
- The SQLTYPE has changed from the previous version for portability in DB2. The values from the previous version (see previous version SQL Reference) will continue to be supported.

Unrecognized and unsupported SQLTYPEs

The values that appear in the SQLTYPE field of the SQLDA are dependent on the level of data type support available at the sender as well as at the receiver of the data. This is particularly important as new data types are added to the product.

New data types may or may not be supported by the sender or receiver of the data and may or may not even be recognized by the sender or receiver of the data. Depending on the situation, the new data type may be returned, or a compatible data type agreed upon by both the sender and receiver of the data may be returned or an error may result.

When the sender and receiver agree to use a compatible data type, the following indicates the mapping that will take place. This mapping will take place when at least one of the sender or the receiver does not support the data type provided. The unsupported data type can be provided by either the application or the database manager.

Data Type	Compatible Data Type
BIGINT	DECIMAL(19, 0)
ROWID[1]	VARCHAR(40) FOR BIT DATA

[1] ROWID is supported by DB2 Universal Database for z/OS Version 8.

Note that no indication is given in the SQLDA that the data type is substituted.

Packed decimal numbers

Packed decimal numbers are stored in a variation of Binary Coded Decimal (BCD) notation. In BCD, each nybble (four bits) represents one decimal digit. For example, 0001 0111 1001 represents 179. Therefore, read a packed decimal value nybble by nybble. Store the value in bytes and then read those bytes in hexadecimal representation to return to decimal. For example, 0001 0111 1001 becomes 00000001 01111001 in binary representation. By reading this number as hexadecimal, it becomes 0179.

The decimal point is determined by the scale. In the case of a DEC(12,5) column, for example, the rightmost 5 digits are to the right of the decimal point.

Sign is indicated by a nybble to the right of the nybbles representing the digits. A positive or negative sign is indicated as follows:

Table 157. Values for Sign Indicator of a Packed Decimal Number

	Representation		
Sign	Binary	Decimal	Hexadecimal
Positive (+)	1100	12	C
Negative (-)	1101	13	D

In summary:

- To store any value, allocate *p*/2+1 bytes, where *p* is precision.
- Assign the nybbles from left to right to represent the value. If a number has an even precision, a leading zero nybble is added. This assignment includes leading (insignificant) and trailing (significant) zero digits.
- The sign nybble will be the second nybble of the last byte.

For example:

Column	Value	Nybbles in Hexadecimal Grouped by Bytes
DEC(8,3)	6574.23	00 65 74 23 0C

Packed decimal numbers

Column	Value	Nybbles in Hexadecimal Grouped by Bytes
DEC(6,2)	-334.02	00 33 40 2D
DEC(7,5)	5.2323	05 23 23 0C
DEC(5,2)	-23.5	02 35 0D

SQLLEN field for decimal

The SQLLEN field contains the precision (first byte) and scale (second byte) of the decimal column. If writing a portable application, the precision and scale bytes should be set individually, versus setting them together as a short integer. This will avoid integer byte reversal problems.

For example, in C:

```
((char *)&(sqlda->sqlvar[i].sqllen))[0] = precision;
((char *)&(sqlda->sqlvar[i].sqllen))[1] = scale;
```

Related reference:

- "CHAR scalar function" in *SQL Reference, Volume 1*

sql_authorizations

This structure is used to return information after a call to the sqluadau API. The data type of all fields is SMALLINT. The first half of the following table contains authorities granted directly to a user. The second half of the table contains authorities granted to the groups to which a user belongs.

Table 158. Fields in the SQL-AUTHORIZATIONS Structure

Field Name	Description
SQL_AUTHORIZATIONS_LEN SQL_SYSADM_AUTH SQL_SYSCTRL_AUTH SQL_SYSMAINT_AUTH SQL_DBADM_AUTH SQL_CREATETAB_AUTH SQL_CREATET_NOT_FENC_AUTH SQL_BINDADD_AUTH SQL_CONNECT_AUTH SQL_IMPLICIT_SCHEMA_AUTH SQL_LOAD_AUTH SQL_SYSADM_GRP_AUTH SQL_SYSCTRL_GRP_AUTH SQL_SYSMAINT_GRP_AUTH SQL_DBADM_GRP_AUTH SQL_CREATETAB_GRP_AUTH SQL_CREATE_NON_FENC_GRP_AUTH SQL_BINDADD_GRP_AUTH SQL_CONNECT_GRP_AUTH SQL_IMPLICIT_SCHEMA_GRP_AUTH SQL_LOAD_GRP_AUTH SQL_CREATE_EXT_RTN_AUTH SQL_CREATE_EXT_RTN_GRP_AUTH SQL_QUIESCE_CONNECT_AUTH SQL_QUIESCE_CONNECT_GRP_AUTH SQL_SECURITY_ADMIN_AUTH SQL_SECURITY_ADMIN_GRP_AUTH SQL_SYSMON_AUTH SQL_SYSMON_GRP_AUTH	Size of structure. SYSADM authority. SYSCTRL authority. SYSMAINT authority. DBADM authority. CREATETAB authority. CREATE_NOT_FENCED authority. BINDADD authority. CONNECT authority. IMPLICIT_SCHEMA authority. LOAD authority. User belongs to a group which holds SYSADM authority. User belongs to a group which holds SYSCTRL authority. User belongs to a group which holds SYSMAINT authority. User belongs to a group which holds DBADM authority. User belongs to a group which holds CREATETAB authority. User belongs to a group which holds CREATE_NOT_FENCED authority. User belongs to a group which holds BINDADD authority. User belongs to a group which holds CONNECT authority. User belongs to a group which holds IMPLICIT_SCHEMA authority. User belongs to a group which holds LOAD authority. CREATE_EXTERNAL_ROUTINE authority. User belongs to a group which holds CREATE_EXTERNAL_ROUTINE authority. QUIESCE CONNECT authority. User belongs to a group which holds QUIESCE CONNECT authority. SECADM authority. User belongs to a group which holds SECADM authority. SYSMON authority. User belongs to a group which holds SYSMON authority.

Note: SYSADM, SYSMAINT, and SYSCTRL are only indirect authorities and cannot be granted directly to the user. They are available only through the groups to which the user belongs.

API and data structure syntax:

```
SQL_STRUCTURE sql_authorizations
{
   short sql_authorizations_len;
   short sql_sysadm_auth;
   short sql_dbadm_auth;
   short sql_createtab_auth;
   short sql_bindadd_auth;
   short sql_connect_auth;
   short sql_sysadm_grp_auth;
   short sql_dbadm_grp_auth;
   short sql_createtab_grp_auth;
   short sql_bindadd_grp_auth;
   short sql_connect_grp_auth;
   short sql_sysctrl_auth;
   short sql_sysctrl_grp_auth;
   short sql_sysmaint_auth;
   short sql_sysmaint_grp_auth;
   short sql_create_not_fenc_auth;
   short sql_create_not_fenc_grp_auth;
   short sql_implicit_schema_auth;
   short sql_implicit_schema_grp_auth;
   short sql_load_auth;
   short sql_load_grp_auth;
   short sql_create_ext_rtn_auth;
   short sql_create_ext_rtn_grp_auth;
   short sql_quiesce_connect_auth;
   short sql_quiesce_connect_grp_auth;
   short sql_security_admin_auth;
   short sql_security_admin_grp_auth;
   short sql_library_admin_auth;
   short sql_library_admin_grp_auth;
   short sql_sysmon_auth;
   short sql_sysmon_grp_auth;
};
```

COBOL Structure:

```
* File: sqlutil.cbl
01 SQL-AUTHORIZATIONS.
    05 SQL-AUTHORIZATIONS-LEN PIC S9(4) COMP-5.
    05 SQL-SYSADM-AUTH        PIC S9(4) COMP-5.
    05 SQL-DBADM-AUTH         PIC S9(4) COMP-5.
    05 SQL-CREATETAB-AUTH     PIC S9(4) COMP-5.
    05 SQL-BINDADD-AUTH       PIC S9(4) COMP-5.
    05 SQL-CONNECT-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSADM-GRP-AUTH    PIC S9(4) COMP-5.
    05 SQL-DBADM-GRP-AUTH     PIC S9(4) COMP-5.
    05 SQL-CREATETAB-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-BINDADD-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-CONNECT-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-AUTH       PIC S9(4) COMP-5.
    05 SQL-SYSCTRL-GRP-AUTH   PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-AUTH      PIC S9(4) COMP-5.
    05 SQL-SYSMAINT-GRP-AUTH  PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-AUTH PIC S9(4) COMP-5.
    05 SQL-CREATE-NOT-FENC-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-AUTH PIC S9(4) COMP-5.
    05 SQL-IMPLICIT-SCHEMA-GRP-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-AUTH PIC S9(4) COMP-5.
    05 SQL-LOAD-GRP-AUTH PIC S9(4) COMP-5.
*
```

Related reference:

- "sqluadau - Get current user's authorities" on page 818

Identifiers

An *identifier* is a token that is used to form a name. An identifier in an SQL statement is either an SQL identifier or a host identifier.

- SQL identifiers

  There are two types of *SQL identifiers*: ordinary and delimited.

  - An *ordinary identifier* is an uppercase letter followed by zero or more characters, each of which is an uppercase letter, a digit, or the underscore character. An ordinary identifier should not be identical to a reserved word.

    Examples

    ```
    WKLYSAL     WKLY_SAL
    ```

  - A *delimited identifier* is a sequence of one or more characters enclosed by double quotation marks. Two consecutive quotation marks are used to represent one quotation mark within the delimited identifier. In this way an identifier can include lowercase letters.

    Examples

```
"WKLY_SAL"     "WKLY SAL"     "UNION"     "wkly_sal"
```

Character conversion of identifiers created on a double-byte code page, but used by an application or database on a multi-byte code page, may require special consideration: After conversion, such identifiers may exceed the length limit for an identifier.

- Host identifiers

  A *host identifier* is a name declared in the host program. The rules for forming a host identifier are the rules of the host language. A host identifier should not be greater than 255 bytes in length and should not begin with SQL or DB2 (in uppercase or lowercase characters).

Naming conventions and implicit object name qualifications

The rules for forming the name of an object depend on the object type. Database object names may be made up of a single identifier, or they may be schema-qualified objects made up of two identifiers. Schema-qualified object names may be specified without the schema name; in such cases, the schema name is implicit.

In dynamic SQL statements, a schema-qualified object name implicitly uses the CURRENT SCHEMA special register value as the qualifier for unqualified object name references. By default it is set to the current authorization ID. If the dynamic SQL statement is contained in a package that exhibits bind, define, or invoke behaviour, the CURRENT SCHEMA special register is not used for qualification. In a bind behaviour package, the package default qualifier is used as the value for implicit qualification of unqualified object references. In a define behaviour package, the authorization ID of the routine definer is used as the value for implicit qualification of unqualified object references within that routine. In an invoke behaviour package, the statement authorization ID in effect when the routine is invoked is used as the value for implicit qualification of unqualified object references within dynamic SQL statements within that routine. For more information, see "Dynamic SQL characteristics at run time" on page 1379.

In static SQL statements, the QUALIFIER precompile/bind option implicitly specifies the qualifier for unqualified database object names. By default, this value is set to the package authorization ID.

The following object names, when used in the context of an SQL procedure, are permitted to use only the characters allowed in an ordinary identifier, even if the names are delimited:

- condition-name
- label
- parameter-name
- procedure-name
- SQL-variable-name
- statement-name

The syntax diagrams use different terms for different types of names. The following list defines these terms.

alias-name	A schema-qualified name that designates an alias.
attribute-name	An identifier that designates an attribute of a structured data type.
authorization-name	An identifier that designates a user or a group:

	• Valid characters are: ′A′ through ′Z′; ′a′ through ′z′; ′0′ through ′9′; ′#′; ′@′; ′$′; ′_′; ′!′; ′%′; ′(′; ′)′; ′{′; ′}′; ′-′; ′.′; and ′^′. • The following characters must be delimited with quotation marks when entered through the command line processor: ′!′; ′%′; ′(′; ′)′; ′{′; ′}′; ′-′; ′.′; and ′^′. • The name must not begin with the characters ′SYS′, ′IBM′, or ′SQL′. • The name must not be: ′ADMINS′, ′GUESTS′, ′LOCAL′, ′PUBLIC′, or ′USERS′. • A delimited authorization ID must not contain lowercase letters.
bufferpool-name	An identifier that designates a buffer pool.
column-name	A qualified or unqualified name that designates a column of a table or view. The qualifier is a table name, a view name, a nickname, or a correlation name.
component-name	An identifier that designates a security label component.
condition-name	An identifier that designates a condition in an SQL procedure.
constraint-name	An identifier that designates a referential constraint, primary key constraint, unique constraint, or a table check constraint.
correlation-name	An identifier that designates a result table.
cursor-name	An identifier that designates an SQL cursor. For host compatibility, a hyphen character may be used in the name.
data-source-name	An identifier that designates a data source. This identifier is the first part of a three-part remote object name.
db-partition-group-name	An identifier that designates a database partition group.
descriptor-name	A colon followed by a host identifier that designates an SQL descriptor area (SQLDA). For the description of a host identifier, see “References to host variables” on page 1387. Note that a descriptor name never includes an indicator variable.
distinct-type-name	A qualified or unqualified name that designates a distinct type. An unqualified distinct type name in an SQL statement is implicitly qualified by the database manager, depending on context.
event-monitor-name	An identifier that designates an event monitor.
function-mapping-name	An identifier that designates a function mapping.
function-name	A qualified or unqualified name that designates a function. An unqualified function name in an SQL

	statement is implicitly qualified by the database manager, depending on context.
group-name	An unqualified identifier that designates a transform group defined for a structured type.
host-variable	A sequence of tokens that designates a host variable. A host variable includes at least one host identifier, explained in "References to host variables" on page 1387.
index-name	A schema-qualified name that designates an index or an index specification.
label	An identifier that designates a label in an SQL procedure.
method-name	An identifier that designates a method. The schema context for a method is determined by the schema of the subject type (or a supertype of the subject type) of the method.
nickname	A schema-qualified name that designates a federated server reference to a table or a view.
package-name	A schema-qualified name that designates a package. If a package has a version ID that is not the empty string, the package name also includes the version ID at the end of the name, in the form: `schema-id.package-id.version-id`.
parameter-name	An identifier that designates a parameter that can be referenced in a procedure, user-defined function, method, or index extension.
partition-name	An identifier that designates a data partition in a partitioned table.
procedure-name	A qualified or unqualified name that designates a procedure. An unqualified procedure name in an SQL statement is implicitly qualified by the database manager, depending on context.
remote-authorization-name	An identifier that designates a data source user. The rules for authorization names vary from data source to data source.
remote-function-name	A name that designates a function registered to a data source database.
remote-object-name	A three-part name that designates a data source table or view, and that identifies the data source in which the table or view resides. The parts of this name are data-source-name, remote-schema-name, and remote-table-name.
remote-schema-name	A name that designates the schema to which a data source table or view belongs. This name is the second part of a three-part remote object name.
remote-table-name	A name that designates a table or view at a data source. This name is the third part of a three-part remote object name.

remote-type-name	A data type supported by a data source database. Do not use the long form for built-in types (use CHAR instead of CHARACTER, for example).
savepoint-name	An identifier that designates a savepoint.
schema-name	An identifier that provides a logical grouping for SQL objects. A schema name used as a qualifier for the name of an object may be implicitly determined: • from the value of the CURRENT SCHEMA special register • from the value of the QUALIFIER precompile/bind option • on the basis of a resolution algorithm that uses the CURRENT PATH special register • on the basis of the schema name for another object in the same SQL statement. To avoid complications, it is recommended that the name SESSION not be used as a schema, except as the schema for declared global temporary tables (which *must* use the schema name SESSION).
security-label-name	An identifier that designates a security label.
security-policy-name	An identifier that designates a security policy.
sequence-name	An identifier that designates a sequence.
server-name	An identifier that designates an application server. In a federated system, the server name also designates the local name of a data source.
specific-name	A qualified or unqualified name that designates a specific name. An unqualified specific name in an SQL statement is implicitly qualified by the database manager, depending on context.
SQL-variable-name	The name of a local variable in an SQL procedure statement. SQL variable names can be used in other SQL statements where a host variable name is allowed. The name can be qualified by the label of the compound statement that declared the SQL variable.
statement-name	An identifier that designates a prepared SQL statement.
supertype-name	A qualified or unqualified name that designates the supertype of a type. An unqualified supertype name in an SQL statement is implicitly qualified by the database manager, depending on context.
table-name	A schema-qualified name that designates a table.
tablespace-name	An identifier that designates a table space.
trigger-name	A schema-qualified name that designates a trigger.
type-mapping-name	An identifier that designates a data type mapping.
type-name	A qualified or unqualified name that designates a

	type. An unqualified type name in an SQL statement is implicitly qualified by the database manager, depending on context.
typed-table-name	A schema-qualified name that designates a typed table.
typed-view-name	A schema-qualified name that designates a typed view.
view-name	A schema-qualified name that designates a view.
wrapper-name	An identifier that designates a wrapper.
xmlschema-name	A qualified or unqualified name that designates an XML schema.
xsrobject-name	A qualified or unqualified name that designates an object in the XML schema repository.

Aliases

A table alias can be thought of as an alternative name for a table or a view. A table or view, therefore, can be referred to in an SQL statement by its name or by a table alias.

An alias can be used wherever a table or a view name can be used. An alias can be created even if the object does not exist (although it must exist by the time a statement referring to it is compiled). It can refer to another alias if no circular or repetitive references are made along the chain of aliases. An alias can only refer to a table, view, or alias within the same database. An alias name cannot be used where a new table or view name is expected, such as in the CREATE TABLE or CREATE VIEW statements; for example, if the alias name PERSONNEL has been created, subsequent statements such as CREATE TABLE PERSONNEL... will return an error.

The option of referring to a table or a view by an alias is not explicitly shown in the syntax diagrams, or mentioned in the descriptions of SQL statements.

A new unqualified alias cannot have the same fully-qualified name as an existing table, view, or alias.

The effect of using an alias in an SQL statement is similar to that of text substitution. The alias, which must be defined by the time that the SQL statement is compiled, is replaced at statement compilation time by the qualified base table or view name. For example, if PBIRD.SALES is an alias for DSPN014.DIST4_SALES_148, then at compilation time:

```
SELECT * FROM PBIRD.SALES
```

effectively becomes

```
SELECT * FROM DSPN014.DIST4_SALES_148
```

In a federated system, the aforementioned uses and restrictions apply, not only to table aliases, but also to aliases for nicknames. Thus, a nickname's alias can be used instead of the nickname in an SQL statement; an alias can be created for a nickname that does not yet exist, provided that the nickname is created before statements that reference the alias are compiled; an alias for a nickname can refer to another alias for that nickname; and so on.

For syntax toleration of applications running under other relational database management systems, SYNONYM can be used in place of ALIAS in the CREATE ALIAS and DROP ALIAS statements.

Authorization IDs and authorization names

An *authorization ID* is a character string that is obtained by the database manager when a connection is established between the database manager and either an application process or a program preparation process. It designates a set of privileges. It may also designate a user or a group of users, but this property is not controlled by the database manager.

Authorization IDs are used by the database manager to provide:
- Authorization checking of SQL statements
- A default value for the QUALIFIER precompile/bind option and the CURRENT SCHEMA special register. The authorization ID is also included in the default CURRENT PATH special register and the FUNCPATH precompile/bind option.

An authorization ID applies to every SQL statement. The authorization ID that applies to a static SQL statement is the authorization ID that is used during program binding. The authorization ID that applies to a dynamic SQL statement is based on the DYNAMICRULES option supplied at bind time, and on the current runtime environment for the package issuing the dynamic SQL statement:
- In a package that has bind behavior, the authorization ID used is the authorization ID of the package owner.
- In a package that has define behavior, the authorization ID used is the authorization ID of the corresponding routine's definer.
- In a package that has run behavior, the authorization ID used is the current authorization ID of the user executing the package.
- In a package that has invoke behavior, the authorization ID used is the authorization ID currently in effect when the routine is invoked. This is called the runtime authorization ID.

For more information, see "Dynamic SQL characteristics at run time" on page 1379.

An *authorization name* specified in an SQL statement should not be confused with the authorization ID of the statement. An authorization name is an identifier that is used within various SQL statements. An authorization name is used in the CREATE SCHEMA statement to designate the owner of the schema. An authorization name is used in the GRANT and REVOKE statements to designate a target of the grant or revoke operation. Granting privileges to *X* means that *X* (or a member of the group *X*) will subsequently be the authorization ID of statements that require those privileges.

Examples:
- Assume that SMITH is the user ID and the authorization ID that the database manager obtained when a connection was established with the application process. The following statement is executed interactively:

  ```
  GRANT SELECT ON TDEPT TO KEENE
  ```

  SMITH is the authorization ID of the statement. Therefore, in a dynamic SQL statement, the default value of the CURRENT SCHEMA special register is SMITH, and in static SQL, the default value of the QUALIFIER precompile/bind option is SMITH. The authority to execute the statement is checked against

SMITH, and SMITH is the *table-name* implicit qualifier based on qualification rules described in "Naming conventions and implicit object name qualifications" on page 1373.

KEENE is an authorization name specified in the statement. KEENE is given the SELECT privilege on SMITH.TDEPT.

- Assume that SMITH has administrative authority and is the authorization ID of the following dynamic SQL statements, with no SET SCHEMA statement issued during the session:

  ```
  DROP TABLE TDEPT
  ```

  Removes the SMITH.TDEPT table.

  ```
  DROP TABLE SMITH.TDEPT
  ```

  Removes the SMITH.TDEPT table.

  ```
  DROP TABLE KEENE.TDEPT
  ```

  Removes the KEENE.TDEPT table. Note that KEENE.TDEPT and SMITH.TDEPT are different tables.

  ```
  CREATE SCHEMA PAYROLL AUTHORIZATION KEENE
  ```

  KEENE is the authorization name specified in the statement that creates a schema called PAYROLL. KEENE is the owner of the schema PAYROLL and is given CREATEIN, ALTERIN, and DROPIN privileges, with the ability to grant them to others.

Dynamic SQL characteristics at run time

The BIND option DYNAMICRULES determines the authorization ID that is used for checking authorization when dynamic SQL statements are processed. In addition, the option also controls other dynamic SQL attributes, such as the implicit qualifier that is used for unqualified object references, and whether certain SQL statements can be invoked dynamically.

The set of values for the authorization ID and other dynamic SQL attributes is called the dynamic SQL statement behavior. The four possible behaviors are run, bind, define, and invoke. As the following table shows, the combination of the value of the DYNAMICRULES BIND option and the runtime environment determines which of the behaviors is used. DYNAMICRULES RUN, which implies run behavior, is the default.

Table 159. How DYNAMICRULES and the runtime environment determine dynamic SQL statement behavior

	Behavior of dynamic SQL statements	
DYNAMICRULES value	**Standalone program environment**	**Routine environment**
BIND	Bind behavior	Bind behavior
RUN	Run behavior	Run behavior
DEFINEBIND	Bind behavior	Define behavior
DEFINERUN	Run behavior	Define behavior
INVOKEBIND	Bind behavior	Invoke behavior
INVOKERUN	Run behavior	Invoke behavior

Run behavior	DB2 uses the authorization ID of the user (the ID that initially connected to DB2) executing the package as the value to be used for authorization

checking of dynamic SQL statements and for the initial value used for implicit qualification of unqualified object references within dynamic SQL statements.

Bind behavior
At run time, DB2 uses all the rules that apply to static SQL for authorization and qualification. It takes the authorization ID of the package owner as the value to be used for authorization checking of dynamic SQL statements, and the package default qualifier for implicit qualification of unqualified object references within dynamic SQL statements.

Define behavior
Define behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES DEFINEBIND or DYNAMICRULES DEFINERUN. DB2 uses the authorization ID of the routine definer (not the routine's package binder) as the value to be used for authorization checking of dynamic SQL statements, and for implicit qualification of unqualified object references within dynamic SQL statements within that routine.

Invoke behavior
Invoke behavior applies only if the dynamic SQL statement is in a package that is run within a routine context, and the package was bound with DYNAMICRULES INVOKEBIND or DYNAMICRULES INVOKERUN. DB2 uses the statement authorization ID in effect when the routine is invoked as the value to be used for authorization checking of dynamic SQL, and for implicit qualification of unqualified object references within dynamic SQL statements within that routine. This is summarized by the following table.

Invoking Environment	ID Used
any static SQL	implicit or explicit value of the OWNER of the package the SQL invoking the routine came from
used in definition of view or trigger	definer of the view or trigger
dynamic SQL from a bind behavior package	implicit or explicit value of the OWNER of the package the SQL invoking the routine came from
dynamic SQL from a run behavior package	ID used to make the initial connection to DB2
dynamic SQL from a define behavior package	definer of the routine that uses the package that the SQL invoking the routine came from
dynamic SQL from an invoke behavior package	the current authorization ID invoking the routine

Restricted statements when run behavior does not apply

When bind, define, or invoke behavior is in effect, you cannot use the following dynamic SQL statements: GRANT, REVOKE, ALTER, CREATE, DROP, COMMENT, RENAME, SET INTEGRITY, SET EVENT MONITOR STATE; or queries that reference a nickname.

Considerations regarding the DYNAMICRULES option

The CURRENT SCHEMA special register cannot be used to qualify unqualified object references within dynamic SQL statements executed from bind, define or invoke behavior packages. This is true even after you issue the SET CURRENT SCHEMA statement to change the CURRENT SCHEMA special register; the register value is changed but not used.

In the event that multiple packages are referenced during a single connection, all dynamic SQL statements prepared by those packages will exhibit the behavior specified by the DYNAMICRULES option for that specific package and the environment in which they are used.

It is important to keep in mind that when a package exhibits bind behavior, the binder of the package should not have any authorities granted that the user of the package should not receive, because a dynamic statement will be using the authorization ID of the package owner. Similarly, when a package exhibits define behavior, the definer of the routine should not have any authorities granted that the user of the package should not receive.

Authorization IDs and statement preparation

If the VALIDATE BIND option is specified at bind time, the privileges required to manipulate tables and views must also exist at bind time. If these privileges or the referenced objects do not exist, and the SQLERROR NOPACKAGE option is in effect, the bind operation will be unsuccessful. If the SQLERROR CONTINUE option is specified, the bind operation will be successful, and any statements in error will be flagged. Any attempt to execute such a statement will result in an error.

If a package is bound with the VALIDATE RUN option, all normal bind processing is completed, but the privileges required to use the tables and views that are referenced in the application need not exist yet. If a required privilege does not exist at bind time, an incremental bind operation is performed whenever the statement is first executed in an application, and all privileges required for the statement must exist. If a required privilege does not exist, execution of the statement is unsuccessful.

Authorization checking at run time is performed using the authorization ID of the package owner.

Column names

The meaning of a *column name* depends on its context. A column name can be used to:

- Declare the name of a column, as in a CREATE TABLE statement.
- Identify a column, as in a CREATE INDEX statement.
- Specify values of the column, as in the following contexts:
  - In a column function, a column name specifies all values of the column in the group or intermediate result table to which the function is applied. For example, MAX(SALARY) applies the function MAX to all values of the column SALARY in a group.
  - In a GROUP BY or ORDER BY clause, a column name specifies all values in the intermediate result table to which the clause is applied. For example, ORDER BY DEPT orders an intermediate result table by the values of the column DEPT.

  - In an expression, a search condition, or a scalar function, a column name specifies a value for each row or group to which the construct is applied. For example, when the search condition CODE = 20 is applied to some row, the value specified by the column name CODE is the value of the column CODE in that row.
- Temporarily rename a column, as in the *correlation-clause* of a *table-reference* in a FROM clause.

Qualified column names

A qualifier for a column name may be a table, view, nickname, alias, or correlation name.

Whether a column name may be qualified depends on its context:

- Depending on the form of the COMMENT ON statement, a single column name may need to be qualified. Multiple column names must be unqualified.
- Where the column name specifies values of the column, it may be qualified at the user’s option.
- In the assignment-clause of an UPDATE statement, it may be qualified at the user’s option.
- In all other contexts, a column name must not be qualified.

Where a qualifier is optional, it can serve two purposes. They are described under “Column name qualifiers to avoid ambiguity” on page 1384 and “Column name qualifiers in correlated references” on page 1385.

Correlation names

A *correlation name* can be defined in the FROM clause of a query and in the first clause of an UPDATE or DELETE statement. For example, the clause FROM X.MYTABLE Z establishes Z as a correlation name for X.MYTABLE.

```
FROM X.MYTABLE Z
```

With Z defined as a correlation name for X.MYTABLE, only Z can be used to qualify a reference to a column of that instance of X.MYTABLE in that SELECT statement.

A correlation name is associated with a table, view, nickname, alias, nested table expression or table function only within the context in which it is defined. Hence, the same correlation name can be defined for different purposes in different statements, or in different clauses of the same statement.

As a qualifier, a correlation name can be used to avoid ambiguity or to establish a correlated reference. It can also be used merely as a shorter name for a table, view, nickname, or alias. In the case of a nested table expression or table function, a correlation name is required to identify the result table. In the example, Z might have been used merely to avoid having to enter X.MYTABLE more than once.

If a correlation name is specified for a table, view, nickname, or alias name, any qualified reference to a column of that instance of the table, view, nickname, or alias must use the correlation name, rather than the table, view, nickname, or alias name. For example, the reference to EMPLOYEE.PROJECT in the following example is incorrect, because a correlation name has been specified for EMPLOYEE:

Example

```
FROM EMPLOYEE E
  WHERE EMPLOYEE.PROJECT='ABC'      * incorrect*
```

The qualified reference to PROJECT should instead use the correlation name, ″E″, as shown below:

```
FROM EMPLOYEE E
  WHERE E.PROJECT='ABC'
```

Names specified in a FROM clause are either *exposed* or *non-exposed*. A table, view, nickname, or alias name is said to be exposed in the FROM clause if a correlation name is not specified. A correlation name is always an exposed name. For example, in the following FROM clause, a correlation name is specified for EMPLOYEE but not for DEPARTMENT, so DEPARTMENT is an exposed name, and EMPLOYEE is not:

```
FROM EMPLOYEE E, DEPARTMENT
```

A table, view, nickname, or alias name that is exposed in a FROM clause may be the same as any other table name, view name or nickname exposed in that FROM clause or any correlation name in the FROM clause. This may result in ambiguous column name references which returns an error (SQLSTATE 42702).

The first two FROM clauses shown below are correct, because each one contains no more than one reference to EMPLOYEE that is exposed:

1. Given the FROM clause:

   ```
   FROM EMPLOYEE E1, EMPLOYEE
   ```

   a qualified reference such as EMPLOYEE.PROJECT denotes a column of the second instance of EMPLOYEE in the FROM clause. A qualified reference to the first instance of EMPLOYEE must use the correlation name “E1” (E1.PROJECT).
2. Given the FROM clause:

   ```
   FROM EMPLOYEE, EMPLOYEE E2
   ```

   a qualified reference such as EMPLOYEE.PROJECT denotes a column of the first instance of EMPLOYEE in the FROM clause. A qualified reference to the second instance of EMPLOYEE must use the correlation name “E2” (E2.PROJECT).
3. Given the FROM clause:

   ```
   FROM EMPLOYEE, EMPLOYEE
   ```

   the two exposed table names included in this clause (EMPLOYEE and EMPLOYEE) are the same. This is allowed, but references to specific column names would be ambiguous (SQLSTATE 42702).
4. Given the following statement:

   ```
   SELECT *
     FROM EMPLOYEE E1, EMPLOYEE E2            * incorrect *
     WHERE EMPLOYEE.PROJECT = 'ABC'
   ```

   the qualified reference EMPLOYEE.PROJECT is incorrect, because both instances of EMPLOYEE in the FROM clause have correlation names. Instead, references to PROJECT must be qualified with either correlation name (E1.PROJECT or E2.PROJECT).
5. Given the FROM clause:

   ```
   FROM EMPLOYEE, X.EMPLOYEE
   ```

   a reference to a column in the second instance of EMPLOYEE must use X.EMPLOYEE (X.EMPLOYEE.PROJECT). If X is the CURRENT SCHEMA special register value in dynamic SQL or the QUALIFIER precompile/bind option in static SQL, then the columns cannot be referenced since any such reference would be ambiguous.

The use of a correlation name in the FROM clause also allows the option of specifying a list of column names to be associated with the columns of the result table. As with a correlation name, these listed column names become the *exposed* names of the columns that must be used for references to the columns throughout the query. If a column name list is specified, then the column names of the underlying table become *non-exposed*.

Given the FROM clause:

```
FROM DEPARTMENT D (NUM,NAME,MGR,ANUM,LOC)
```

a qualified reference such as D.NUM denotes the first column of the DEPARTMENT table that is defined in the table as DEPTNO. A reference to D.DEPTNO using this FROM clause is incorrect since the column name DEPTNO is a non-exposed column name.

Column name qualifiers to avoid ambiguity

In the context of a function, a GROUP BY clause, ORDER BY clause, an expression, or a search condition, a column name refers to values of a column in some table, view, nickname, nested table expression or table function. The tables, views, nicknames, nested table expressions and table functions that might contain the column are called the *object tables* of the context. Two or more object tables might contain columns with the same name; one reason for qualifying a column name is to designate the table from which the column comes. Qualifiers for column names are also useful in SQL procedures to distinguish column names from SQL variable names used in SQL statements.

A nested table expression or table function will consider *table-references* that precede it in the FROM clause as object tables. The *table-references* that follow are not considered as object tables.

Table designators: A qualifier that designates a specific object table is called a *table designator*. The clause that identifies the object tables also establishes the table designators for them. For example, the object tables of an expression in a SELECT clause are named in the FROM clause that follows it:

```
SELECT CORZ.COLA, OWNY.MYTABLE.COLA
  FROM OWNX.MYTABLE CORZ, OWNY.MYTABLE
```

Table designators in the FROM clause are established as follows:

- A name that follows a table, view, nickname, alias, nested table expression or table function is both a correlation name and a table designator. Thus, CORZ is a table designator. CORZ is used to qualify the first column name in the select list.
- An exposed table, view name, nickname or alias is a table designator. Thus, OWNY.MYTABLE is a table designator. OWNY.MYTABLE is used to qualify the second column name in the select list.

Each table designator should be unique within a particular FROM clause to avoid the possibility of ambiguous references to columns.

Avoiding undefined or ambiguous references: When a column name refers to values of a column, exactly one object table must include a column with that name. The following situations are considered errors:

- No object table contains a column with the specified name. The reference is undefined.
- The column name is qualified by a table designator, but the table designated does not include a column with the specified name. Again the reference is undefined.
- The name is unqualified, and more than one object table includes a column with that name. The reference is ambiguous.
- The column name is qualified by a table designator, but the table designated is not unique in the FROM clause and both occurrences of the designated table include the column. The reference is ambiguous.
- The column name is in a nested table expression which is not preceded by the TABLE keyword or in a table function or nested table expression that is the right operand of a right outer join or a full outer join and the column name does not refer to a column of a *table-reference* within the nested table expression's fullselect. The reference is undefined.

Avoid ambiguous references by qualifying a column name with a uniquely defined table designator. If the column is contained in several object tables with different names, the table names can be used as designators. Ambiguous references can also be avoided without the use of the table designator by giving unique names to the columns of one of the object tables using the column name list following the correlation name.

When qualifying a column with the exposed table name form of a table designator, either the qualified or unqualified form of the exposed table name may be used. However, the qualifier used and the table used must be the same after fully qualifying the table name, view name or nickname and the table designator.

1. If the authorization ID of the statement is CORPDATA:

   ```
   SELECT CORPDATA.EMPLOYEE.WORKDEPT
     FROM EMPLOYEE
   ```

   is a valid statement.
2. If the authorization ID of the statement is REGION:

   ```
   SELECT CORPDATA.EMPLOYEE.WORKDEPT
     FROM EMPLOYEE                        * incorrect *
   ```

   is invalid, because EMPLOYEE represents the table REGION.EMPLOYEE, but the qualifier for WORKDEPT represents a different table, CORPDATA.EMPLOYEE.

Column name qualifiers in correlated references

A *fullselect* is a form of a query that may be used as a component of various SQL statements. A fullselect used within a search condition of any statement is called a *subquery*. A fullselect used to retrieve a single value as an expression within a statement is called a *scalar fullselect* or *scalar subquery*. A fullselect used in the FROM clause of a query is called a *nested table expression*. Subqueries in search conditions, scalar subqueries and nested table expressions are referred to as subqueries through the remainder of this topic.

A subquery may include subqueries of its own, and these may, in turn, include subqueries. Thus an SQL statement may contain a hierarchy of subqueries. Those elements of the hierarchy that contain subqueries are said to be at a higher level than the subqueries they contain.

Every element of the hierarchy contains one or more table designators. A subquery can reference not only the columns of the tables identified at its own level in the hierarchy, but also the columns of the tables identified previously in the hierarchy, back to the highest level of the hierarchy. A reference to a column of a table identified at a higher level is called a *correlated reference*.

For compatibility with existing standards for SQL, both qualified and unqualified column names are allowed as correlated references. However, it is good practice to qualify all column references used in subqueries; otherwise, identical column names may lead to unintended results. For example, if a table in a hierarchy is altered to contain the same column name as the correlated reference and the statement is prepared again, the reference will apply to the altered table.

When a column name in a subquery is qualified, each level of the hierarchy is searched, starting at the same subquery as the qualified column name appears and continuing to the higher levels of the hierarchy until a table designator that matches the qualifier is found. Once found, it is verified that the table contains the given column. If the table is found at a higher level than the level containing column name, then it is a correlated reference to the level where the table designator was found. A nested table expression must be preceded with the optional TABLE keyword in order to search the hierarchy above the fullselect of the nested table expression.

When the column name in a subquery is not qualified, the tables referenced at each level of the hierarchy are searched, starting at the same subquery where the column name appears and continuing to higher levels of the hierarchy, until a match for the column name is found. If the column is found in a table at a higher level than the level containing column name, then it is a correlated reference to the level where the table containing the column was found. If the column name is found in more than one table at a particular level, the reference is ambiguous and considered an error.

In either case, T, used in the following example, refers to the table designator that contains column C. A column name, T.C (where T represents either an implicit or an explicit qualifier), is a correlated reference if, and only if, these conditions are met:

- T.C is used in an expression of a subquery.
- T does not designate a table used in the from clause of the subquery.
- T designates a table used at a higher level of the hierarchy that contains the subquery.

Since the same table, view or nickname can be identified at many levels, unique correlation names are recommended as table designators. If T is used to designate a table at more than one level (T is the table name itself or is a duplicate correlation name), T.C refers to the level where T is used that most directly contains the subquery that includes T.C. If a correlation to a higher level is needed, a unique correlation name must be used.

The correlated reference T.C identifies a value of C in a row or group of T to which two search conditions are being applied: condition 1 in the subquery, and condition 2 at some higher level. If condition 2 is used in a WHERE clause, the subquery is evaluated for each row to which condition 2 is applied. If condition 2 is used in a HAVING clause, the subquery is evaluated for each group to which condition 2 is applied.

For example, in the following statement, the correlated reference X.WORKDEPT (in the last line) refers to the value of WORKDEPT in table EMPLOYEE at the level of the first FROM clause. (That clause establishes X as a correlation name for EMPLOYEE.) The statement lists employees who make less than the average salary for their department.

```
SELECT EMPNO, LASTNAME, WORKDEPT
  FROM EMPLOYEE X
  WHERE SALARY < (SELECT AVG(SALARY)
                    FROM EMPLOYEE
                    WHERE WORKDEPT = X.WORKDEPT)
```

The next example uses THIS as a correlation name. The statement deletes rows for departments that have no employees.

```
DELETE FROM DEPARTMENT THIS
   WHERE NOT EXISTS(SELECT *
                      FROM EMPLOYEE
                      WHERE WORKDEPT = THIS.DEPTNO)
```

References to host variables

A *host variable* is either:

- A variable in a host language such as a C variable, a C++ variable, a COBOL data item, a FORTRAN variable, or a Java variable

or:

- A host language construct that was generated by an SQL precompiler from a variable declared using SQL extensions

that is referenced in an SQL statement. Host variables are either directly defined by statements in the host language or are indirectly defined using SQL extensions.

A host variable in an SQL statement must identify a host variable described in the program according to the rules for declaring host variables.

All host variables used in an SQL statement must be declared in an SQL DECLARE section in all host languages except REXX. No variables may be declared outside an SQL DECLARE section with names identical to variables declared inside an SQL DECLARE section. An SQL DECLARE section begins with BEGIN DECLARE SECTION and ends with END DECLARE SECTION.

The meta-variable *host-variable*, as used in the syntax diagrams, shows a reference to a host variable. A host-variable in the VALUES INTO clause or the INTO clause of a FETCH or a SELECT INTO statement, identifies a host variable to which a value from a column of a row or an expression is assigned. In all other contexts a host-variable specifies a value to be passed to the database manager from the application program.

Host variables in dynamic SQL

In dynamic SQL statements, parameter markers are used instead of host variables. A parameter marker is a question mark (?) representing a position in a dynamic SQL statement where the application will provide a value; that is, where a host variable would be found if the statement string were a static SQL statement. The following example shows a static SQL statement using host variables:

```
INSERT INTO DEPARTMENT
  VALUES (:hv_deptno, :hv_deptname, :hv_mgrno, :hv_admrdept)
```

This example shows a dynamic SQL statement using parameter markers:

```
INSERT INTO DEPARTMENT VALUES (?, ?, ?, ?)
```

Host variables in dynamic SQL

The meta-variable *host-variable* in syntax diagrams can generally be expanded to:

```
►►──:host-identifier──┬────────────────────────────────┬──────────────►◄
                      │ ┌─INDICATOR─┐                  │
                      └─┴───────────┴─:host-identifier─┘
```

Each *host-identifier* must be declared in the source program. The variable designated by the second host-identifier must have a data type of small integer.

The first host-identifier designates the *main variable*. Depending on the operation, it either provides a value to the database manager or is provided a value from the database manager. An input host variable provides a value in the runtime application code page. An output host variable is provided a value that, if necessary, is converted to the runtime application code page when the data is copied to the output application variable. A given host variable can serve as both an input and an output variable in the same program.

The second host-identifier designates its *indicator variable*. The purposes of the indicator variable are to:

- Specify the null value. A negative value of the indicator variable specifies the null value. A value of -2 indicates a numeric conversion or arithmetic expression error occurred in deriving the result
- Record the original length of a truncated string (if the source of the value is not a large object type)
- Record the seconds portion of a time if the time is truncated on assignment to a host variable.

For example, if :HV1:HV2 is used to specify an insert or update value, and if HV2 is negative, the value specified is the null value. If HV2 is not negative the value specified is the value of HV1.

Similarly, if :HV1:HV2 is specified in a VALUES INTO clause or in a FETCH or SELECT INTO statement, and if the value returned is null, HV1 is not changed, and HV2 is set to a negative value. If the database is configured with DFT_SQLMATHWARN yes (or was during binding of a static SQL statement), HV2 could be -2. If HV2 is -2, a value for HV1 could not be returned because of an error converting to the numeric type of HV1, or an error evaluating an arithmetic expression that is used to determine the value for HV1. When accessing a database with a client version earlier than DB2 Universal Database Version 5, HV2 will be -1 for arithmetic exceptions. If the value returned is not null, that value is assigned to HV1 and HV2 is set to zero (unless the assignment to HV1 requires string truncation of a non-LOB string; in which case HV2 is set to the original length of the string). If an assignment requires truncation of the seconds part of a time, HV2 is set to the number of seconds.

If the second host identifier is omitted, the host-variable does not have an indicator variable. The value specified by the host-variable reference :HV1 is always the value of HV1, and null values cannot be assigned to the variable. Thus, this form should not be used in an INTO clause unless the corresponding column cannot contain null values. If this form is used and the column contains nulls, the database manager will generate an error at run time.

An SQL statement that references host variables must be within the scope of the declaration of those host variables. For host variables referenced in the SELECT statement of a cursor, that rule applies to the OPEN statement rather than to the DECLARE CURSOR statement.

Example: Using the PROJECT table, set the host variable PNAME (VARCHAR(26)) to the project name (PROJNAME), the host variable STAFF (dec(5,2)) to the mean staffing level (PRSTAFF), and the host variable MAJPROJ (char(6)) to the major project (MAJPROJ) for project (PROJNO) 'IF1000'. Columns PRSTAFF and MAJPROJ may contain null values, so provide indicator variables STAFF_IND (smallint) and MAJPROJ_IND (smallint).

```
SELECT PROJNAME, PRSTAFF, MAJPROJ
  INTO :PNAME, :STAFF :STAFF_IND, :MAJPROJ :MAJPROJ_IND
  FROM PROJECT
  WHERE PROJNO = 'IF1000'
```

MBCS Considerations: Whether multi-byte characters can be used in a host variable name depends on the host language.

References to BLOB, CLOB, and DBCLOB host variables

Regular BLOB, CLOB, and DBCLOB variables, LOB locator variables (see "References to locator variables"), and LOB file reference variables (see "References to BLOB, CLOB, and DBCLOB file reference variables" on page 1390) can be defined in all host languages. Where LOBs are allowed, the term *host-variable* in a syntax diagram can refer to a regular host variable, a locator variable, or a file reference variable. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable. In the case of REXX, LOBs are mapped to strings.

It is sometimes possible to define a large enough variable to hold an entire large object value. If this is true and if there is no performance benefit to be gained by deferred transfer of data from the server, a locator is not needed. However, since host language or space restrictions will often dictate against storing an entire large object in temporary storage at one time and/or because of performance benefit, a large object may be referenced via a locator and portions of that object may be selected into or updated from host variables that contain only a portion of the large object at one time.

References to locator variables

A *locator variable* is a host variable that contains the locator representing a LOB value on the application server.

A locator variable in an SQL statement must identify a locator variable described in the program according to the rules for declaring locator variables. This is always indirectly through an SQL statement.

The term locator variable, as used in the syntax diagrams, shows a reference to a locator variable. The meta-variable *locator-variable* can be expanded to include a *host-identifier* the same as that for *host-variable*.

As with all other host variables, a large object locator variable may have an associated indicator variable. Indicator variables for large object locator host variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the locator host variable is unchanged. This means a locator can never point to a null value.

If a locator-variable that does not currently represent any value is referenced, an error is raised (SQLSTATE 0F001).

At transaction commit, or any transaction termination, all locators acquired by that transaction are released.

References to BLOB, CLOB, and DBCLOB file reference variables

BLOB, CLOB, and DBCLOB file reference variables are used for direct file input and output for LOBs, and can be defined in all host languages. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable. In the case of REXX, LOBs are mapped to strings.

A file reference variable represents (rather than contains) the file, just as a LOB locator represents, rather than contains, the LOB bytes. Database queries, updates and inserts may use file reference variables to store or to retrieve single column values.

A file reference variable has the following properties:

Data Type
: BLOB, CLOB, or DBCLOB. This property is specified when the variable is declared.

Direction
: This must be specified by the application program at run time (as part of the File Options value). The direction is one of:
- Input (used as a source of data on an EXECUTE statement, an OPEN statement, an UPDATE statement, an INSERT statement, or a DELETE statement).
- Output (used as the target of data on a FETCH statement or a SELECT INTO statement).

File name
: This must be specified by the application program at run time. It is one of:
- The complete path name of the file (which is advised).
- A relative file name. If a relative file name is provided, it is appended to the current path of the client process.

Within an application, a file should only be referenced in one file reference variable.

File Name Length
: This must be specified by the application program at run time. It is the length of the file name (in bytes).

File Options
: An application must assign one of a number of options to a file reference variable before it makes use of that variable. Options are set by an INTEGER value in a field in the file reference variable structure. One of the following values must be specified for each file reference variable:
- Input (from client to server)

SQL_FILE_READ
: This is a regular file that can be opened, read and closed. (The option is SQL-FILE-READ in COBOL, sql_file_read in FORTRAN, and READ in REXX.)

- Output (from server to client)

SQL_FILE_CREATE
: Create a new file. If the file already exists, an error is returned. (The option is SQL-FILE-CREATE in COBOL, sql_file_create in FORTRAN, and CREATE in REXX.)

SQL_FILE_OVERWRITE (Overwrite)
: If an existing file with the specified name exists, it is overwritten; otherwise a new file is created. (The option is SQL-FILE-OVERWRITE in COBOL, sql_file_overwrite in FORTRAN, and OVERWRITE in REXX.)

SQL_FILE_APPEND
: If an existing file with the specified name exists, the output is appended to it; otherwise a new file is created. (The option is SQL-FILE-APPEND in COBOL, sql_file_append in FORTRAN, and APPEND in REXX.)

Data Length
: This is unused on input. On output, the implementation sets the data length to the length of the new data written to the file. The length is in bytes.

As with all other host variables, a file reference variable may have an associated indicator variable.

Example of an output file reference variable (in C): Given a declare section coded as:

```
EXEC SQL BEGIN DECLARE SECTION
   SQL TYPE IS CLOB_FILE  hv_text_file;
   char  hv_patent_title[64];
EXEC SQL END DECLARE SECTION
```

Following preprocessing this would be:

```
EXEC SQL BEGIN DECLARE SECTION
   /* SQL TYPE IS CLOB_FILE  hv_text_file; */
   struct {
       unsigned long  name_length; //  File Name Length
       unsigned long  data_length; //  Data Length
       unsigned long  file_options; // File Options
```

```
        char           name[255];   // File Name
     } hv_text_file;
     char  hv_patent_title[64];
  EXEC SQL END DECLARE SECTION
```

Then, the following code can be used to select from a CLOB column in the database into a new file referenced by :hv_text_file.

```
  strcpy(hv_text_file.name, "/u/gainer/papers/sigmod.94");
  hv_text_file.name_length = strlen("/u/gainer/papers/sigmod.94");
  hv_text_file.file_options = SQL_FILE_CREATE;

  EXEC SQL SELECT content INTO :hv_text_file from papers
       WHERE TITLE = 'The Relational Theory behind Juggling';
```

Example of an input file reference variable (in C): Given the same declare section as above, the following code can be used to insert the data from a regular file referenced by :hv_text_file into a CLOB column.

```
  strcpy(hv_text_file.name, "/u/gainer/patents/chips.13");
  hv_text_file.name_length = strlen("/u/gainer/patents/chips.13");
  hv_text_file.file_options = SQL_FILE_READ:
  strcpy(:hv_patent_title, "A Method for Pipelining Chip Consumption");

  EXEC SQL INSERT INTO patents( title, text )
           VALUES(:hv_patent_title, :hv_text_file);
```

References to structured type host variables

Structured type variables can be defined in all host languages except FORTRAN, REXX, and Java. Since these are not native data types, SQL extensions are used and the precompilers generate the host language constructs necessary to represent each variable.

As with all other host variables, a structured type variable may have an associated indicator variable. Indicator variables for structured type host variables behave in the same way as indicator variables for other data types. When a null value is returned from the database, the indicator variable is set and the structured type host variable is unchanged.

The actual host variable for a structured type is defined as a built-in data type. The built-in data type associated with the structured type must be assignable:

- from the result of the FROM SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command; and
- to the parameter of the TO SQL transform function for the structured type as defined by the specified TRANSFORM GROUP option of the precompile command.

If using a parameter marker instead of a host variable, the appropriate parameter type characteristics must be specified in the SQLDA. This requires a ″doubled″ set of SQLVAR structures in the SQLDA, and the SQLDATATYPE_NAME field of the secondary SQLVAR must be filled with the schema and type name of the structured type. If the schema is omitted in the SQLDA structure, an error results (SQLSTATE 07002).

Example: Define the host variables *hv_poly* and *hv_point* (of type POLYGON, using built-in type BLOB(1048576)) in a C program.

```
EXEC SQL BEGIN DECLARE SECTION;
      static SQL
         TYPE IS POLYGON AS BLOB(1M)
         hv_poly, hv_point;
EXEC SQL END DECLARE SECTION;
```

Related reference:

- “Japanese and traditional-Chinese extended UNIX code (EUC) considerations” in *SQL Reference, Volume 1*
- “Large objects (LOBs)” in *SQL Reference, Volume 1*
- “Reserved schema names and reserved words” on page 29
- “SQL and XQuery limits” in *SQL Reference, Volume 1*
- “CREATE ALIAS statement” in *SQL Reference, Volume 2*
- “PREPARE statement” in *SQL Reference, Volume 2*
- “SET SCHEMA ” on page 1209
- “SQL queries” on page 1210
- “SQLDA (SQL descriptor area)” on page 1361

Example

Part 14. Security plug-ins

Only the default IBM-supplied operating-system based authentication and group plug-ins are supported in Common Criteria compliant environments. User-written or third-party plug-ins are not supported. In addition, Kerberos-based authorization is not supported. The sections that follow are for informational purposes only.

Chapter 44. An overview of security plug-ins

Security plug-ins

Authentication in DB2 is done using *security plug-ins*. A security plug-in is a dynamicaly-loadable library that provides authentication security services. DB2 provides the following types of plug-ins:

- Group retrieval plug-in: retrieves group membership information for a given user.
- Client authentication plug-in: manages authentication on a DB2 client.
- Server authentication plug-in: manages authentication on a DB2 server.

DB2 supports two mechanisms for plug-in authentication:

User ID/password authentication

This involves authentication using a user ID and password. The following authentication types are implemented using user ID/password authentication plug-ins:

- CLIENT
- SERVER
- SERVER_ENCRYPT
- DATA_ENCRYPT
- DATA_ENCRYPT_CMP

These authentication types determine how and where authentication of a user occurs. The authentication type used depends on the authentication type specified by the *authentication* database manager configuration parameter. If the SRVCON_AUTH parameter is specified it takes precedence over AUTHENTICATION when dealing with connect or attach operations.

GSS-API authentication

GSS-API is formally known as *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Kerberos authentication is also implemented using GSS-API. The following authentication types are implemented using GSS-API authentication plug-ins:

- KERBEROS
- GSSPLUGIN
- KRB_SERVER_ENCRYPT
- GSS_SERVER_ENCRYPT

KRB_SERVER_ENCRYPT and GSS_SERVER_ENCRYPT support both GSS-API authentication and user ID/password authentication; however, GSS-API authentication is the preferred authentication type.

Note: Authentication types determine how and where a user is authenticated. To use a particular authentication type, update the authentication database manager configuration parameter.

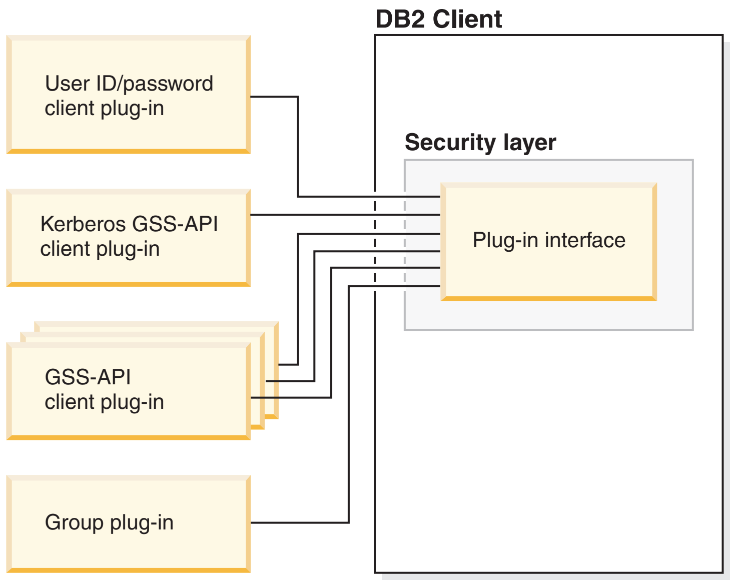
Each of the plug-ins can be used independently or in conjunction with one or more of the other plug-ins. For example, you might only use a server authentication plug-in and assume the DB2 defaults for client and group authentication. Alternatively, you might have only a group or client authentication plug-in. The only situation where both a client and server plug-in are required is for GSS-API authentication plug-ins.

In DB2 Database for Linux, UNIX, and Windows, the default behavior is to use a user ID/password plug-in that implements an operating-system-level mechanism for authentication. In all previous releases of DB2, the default behavior is to directly use operating-system-level authentication without a plug-in implementation. In DB2 Database for Linux, UNIX, and Windows, client-side Kerberos support is available on Solaris, AIX, Windows, and IA32 Linux operating systems; however, it is only enabled by default on Windows.

DB2 database systems include sets of plug-ins for group retrieval, user ID/password authentication, and for Kerberos authentication. With the security plug-in architecture you can customize DB2 client and server authentication behavior by either developing your own plug-ins, or buying plug-ins from a third party.

Deployment of security plug-ins on DB2 clients:

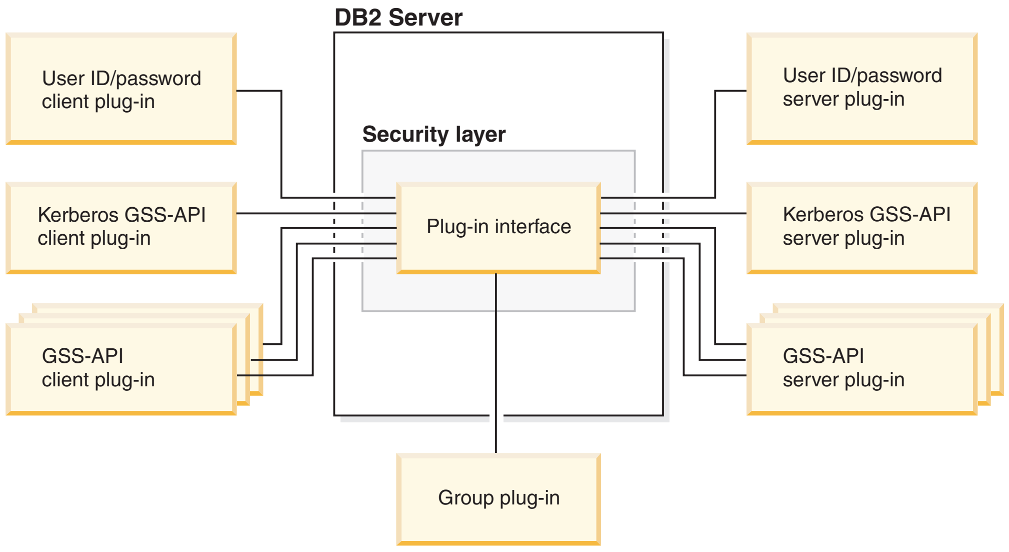
DB2 clients can support one group plug-in, one user ID/password authentication plug-in, and will negotiate with the DB2 server for a particular GSS-API plug-in. This negotiation consists of a scan by the client of the DB2 server's list of implemented GSS-API plug-ins for the first authentication plug-in name that matches an authentication plug-in implemented on the client. The server's list of plug-ins is specified in the *srvcon_gssplugin_list* database manager configuration parameter value, which contains the names of all of the plug-ins that are implemented on the server. The following figure portrays the security plug-in infrastructure on a DB2 client.



Deployment of security plug-ins on DB2 servers:

DB2 servers can support one group plug-in, one user ID/password authentication plug-in, and multiple GSS-API plug-ins. The multiple GSS-API plug-ins are specified in the *srvcon_gssplugin_list* database manager configuration parameter value as a list. Only one GSS-API plug-in in this list can be a Kerberos plug-in.

In addition to server-side security plug-ins, you might also need to deploy client authorization plug-ins on your database server. When you run instance-level operations like `db2start` and `db2trc`, DB2 performs authorization checking for these operations using client authentication plug-ins. Therefore, you should install the client authentication plug-in that corresponds to the server plug-in that is specified by the *authentication* database manager configuration parameter. There is a main distinction between *authentication* and *srvcon_auth*. Specifically, they could be set to different values to cause one mechanism to be used to authenticate database connections and another mechanism to be used for local authorization. The most common usage is *srvcon_auth* set as GSSPLUGIN and *authentication* set as SERVER. If you do not use client authentication plug-ins on the database server, instance level operations such as `db2start` will fail. For example, if the authentication type is SERVER and no user-supplied client plug-in is used, DB2 will use the IBM-shipped default client operating-system plug-in. The following figure portrays the security plug-in infrastructure on a DB2 server.


DB2 Server
User ID/password client plug-in
Kerberos GSS-API client plug-in
GSS-API client plug-in
Security layer
Plug-in interface
User ID/password server plug-in
Kerberos GSS-API server plug-in
GSS-API server plug-in
Group plug-in

Note: The integrity of your DB2 database system installation can be compromised if the deployment of security plug-ins are not adequately coded, reviewed, and tested. DB2 database systems take precautions against many common types of failures, but it cannot guarantee complete integrity when user-written security plug-ins are deployed.

Enabling security plug-ins:

The system administrator can specify the names of the plug-ins to use for each authentication mechanism by updating certain plug-in-related database manager configuration parameters. If these parameters are null, they will default to the DB2-supplied plug-ins for group retrieval, user ID/password management, or Kerberos (if authentication is set to Kerberos -- on the server). DB2 does not provide a default GSS-API plug-in. Therefore, if system administrators specify an authentication type of GSSPLUGIN in *authentication* parameter, they must also specify a GSS-API authentication plug-in in *srvcon_gssplugin_list*.

How DB2 loads security plug-ins:

All of the supported plug-ins identified by the database manager configuration parameters are loaded when the database manager starts.

The DB2 client will load a plug-in appropriate for the security mechanism negotiated with the server during connect or attach operations. It is possible that a client application can cause multiple security plug-ins to be concurrently loaded and used. This situation can occur, for example, in a threaded program that has concurrent connections to different databases from different instances.

Actions other than connect or attach operations require authorization (such as updating the database manager configuration, starting and stopping the database manager, turning DB2 trace on and off) as well. For such actions, the DB2 client program will load a plug-in specified in another database manager configuration parameter. If *authentication* is set to GSSPLUGIN, DB2 database manager will use the plug-in specified by *local_gssplugin*. If *authentication* is set to KERBEROS, DB2 database manager will use the plug-in specified by *clnt_krb_plugin*. Otherwise, DB2 database manager will use the plug-in specified by *clnt_pw_plugin*.

Security plug-ins APIs can be called from either an IPv4 platform or an IPv6 platform. An IPv4 address is a 32-bit address which has a readable form a.b.c.d, where each of a-d represents a decimal number from 0-255. An IPv6 address is a 128 bit address of the form a:b:c:d:e:f:g:h, where each of a-h represents 4 hex digits.

Developing security plug-ins:

If you are developing a security plug-in, you need to implement the standard authentication functions that DB2 database manager will use. If you are using your own customized security plug-in, you can use a user ID of up to 255 characters on a connect statement issued through the CLP or a dynamic SQL statement. For the available types of plug-ins, the functionality you will need to implement is as follows:

Group retrieval
Gets the list of groups to which a user belongs.

User ID/password authentication
- Identifies the default security context (client only).
- Validates and optionally changes a password.
- Determines if a given string represents a valid user (server only).
- Modifies the user ID or password provided on the client before it is sent to the server (client only).
- Returns the DB2 authorization ID associated with a given user.

GSS-API authentication
- Implements the required GSS-API functions.
- Identifies the default security context (client only).
- Generates initial credentials based on a user ID and password and optionally changes password (client only).
- Creates and accepts security tickets.
- Returns the DB2 authorization ID associated with a given GSS-API security context.

Related concepts:
- "DB2 database system plug-ins for customizing database management" in *Administrative API Reference*

- "How DB2 loads security plug-ins" on page 1415
- "Security plug-in APIs" on page 1425
- "Security plug-in library locations" on page 1401
- "Authentication methods for your server" on page 89

Related reference:

- "Security plug-in samples" in *Samples Topics*
- "authentication - Authentication type " on page 1497
- "clnt_krb_plugin - Client Kerberos plug-in " on page 1467
- "clnt_pw_plugin - Client userid-password plug-in " on page 1467
- "local_gssplugin - GSS API plug-in used for local instance level authorization " on page 1468
- "srvcon_auth - Authentication type for incoming connections at the server " on page 1469
- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server " on page 1469

Security plug-in library locations

After you acquire your security plug-ins (by developing them yourself, or purchasing them from a third party), copy them to specific locations on your database server.

DB2 clients looks for client-side user authentication plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/client`
- UNIX 64-bit: `$DB2PATH/security64/plugin/client`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\`*`instance name`*`\client`

Note: On Windows-based platforms, the subdirectories *instance name* and *client* are not created automatically. The instance owner has to manually create them.

The DB2 database manager looks for server-side user authentication plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/server`
- UNIX 64-bit: `$DB2PATH/security64/plugin/server`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\`*`instance name`*`\server`

Note: On Windows-based platforms, the subdirectories *instance name* and *server* are not created automatically. The instance owner has to manually create them.

The DB2 database manager looks for group plug-ins in the following directory:

- UNIX 32-bit: `$DB2PATH/security32/plugin/group`
- UNIX 64-bit: `$DB2PATH/security64/plugin/group`
- WINDOWS 32-bit and 64-bit: `$DB2PATH\security\plugin\`*`instance name`*`\group`

Note: On Windows-based platforms, the subdirectories *instance name* and *group* are not created automatically. The instance owner has to manually create them.

Related concepts:

- "How DB2 loads security plug-ins" on page 1415
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397

Related tasks:
- "Deploying a group retrieval plug-in" on page 1407
- "Deploying a GSS-API plug-in" on page 1410
- "Deploying a Kerberos plug-in" on page 1411
- "Deploying a user ID/password plug-in" on page 1408

Security plug-in naming conventions

Security plug-in libraries must have a platform-specific file name extension. Security plug-in libraries written in C or C++ must have a platform-specific file name extension."
- Windows: `.dll`
- AIX: `.a or .so`, and if both extensions exist, `.a` extension is used.
- Linux, HP IPF and Solaris Operating Environment: `.so`
- HPUX on PA-RISC: `.sl or .so`, and if both extensions exist, `.sl` extension is used.

Note: Users can also develop security plug-ins with the DB2 Universal JDBC Driver.

For example, assume you have a security plug-in library called `MyPlugin`. For each supported operating system, the appropriate library file name follows:
- Windows 32-bit: MyPlugin.dll
- Windows 64-bit: MyPlugin64.dll
- AIX 32 or 64-bit: MyPlugin.a or MyPlugin.so
- SUN 32 or 64-bit, Linux 32 or 64 bit, HP 32 or 64 bit on IPF: MyPlugin.so
- HP-UX 32 or 64-bit on PA-RISC: MyPlugin.sl or MyPlugin.so

Note: The suffix "64" is only required on the library name for 64-bit Windows security plug-ins.

When you update the database manager configuration with the name of a security plug-in, use the full name of the library without the "64" suffix and omit both the file extension and any qualified path portion of the name. Regardless of the operating system, a security plug-in library called `MyPlugin` would be registered as follows:

```
UPDATE DBM CFG USING CLNT_PW_PLUGIN MyPlugin
```

The security plug-in name is case sensitive, and must exactly match the library name. DB2 database systems use the value from the relevant database manager configuration parameter to assemble the library path, and then uses the library path to load the security plug-in library.

To avoid security plug-in name conflicts, you should name the plug-in using the authentication method used, and an identifying symbol of the firm that wrote the plug-in. For instance, if the company `Foo, Inc.` wrote a plug-in implementing the authentication method `FOOsomemethod`, the plug-in could have a name like `FOOsomemethod.dll`.

The maximum length of a plug-in name (not including the file extension and the ″64″ suffix) is limited to 32 bytes. There is no maximum number of plug-ins supported by the database server, but the maximum length of the comma-separated list of plug-ins in the database manager configuration is 255 bytes. Two defines located in the include file `sqlenv.h` identifes these two limits:

```
#define SQL_PLUGIN_NAME_SZ     32     /* plug-in name */
#define SQL_SRVCON_GSSPLUGIN_LIST_SZ 255 /* GSS API plug-in list */
```

The security plug-in library files must have the following file permissions:

- Owned by the instance owner.
- Readable by all users on the system.
- Executable by all users on the system.

Related concepts:

- "Configuration parameters" on page 1475
- "Security plug-in library locations" on page 1401
- "Security plug-ins" on page 1397

Related tasks:

- "Configuring DB2 with configuration parameters" on page 1493
- "Deploying a group retrieval plug-in" on page 1407
- "Deploying a GSS-API plug-in" on page 1410
- "Deploying a Kerberos plug-in" on page 1411
- "Deploying a user ID/password plug-in" on page 1408

Related reference:

- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Security plug-in support for two-part user IDs

DB2 database for Windows supports the use of two-part user IDs, and the mapping of two-part user IDs to two-part authorization IDs.

For example, consider a Windows operating system two-part user ID composed of a domain and user ID such as: `MEDWAY\pieter`. In this example, `MEDWAY` is a domain and `pieter` is the user name. In DB2 database systems, you can specify whether this two-part user ID should be mapped to either a one-part authorization ID or a two-part authorization ID.

The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior. By default, both one-part user IDs and two-part user IDs map to one-part authorization IDs. The mapping of a two-part user ID to a two-part authorization ID is supported, but is not the default behavior. Prior to DB2 UDB, version 8.2, you could only have a one-part user ID that mapped to a one-part authorization ID.

The default mapping of a two-part user ID to a one-part user ID allows a user to connect to the database using:

```
db2 connect to db user MEDWAY\pieter using pw
```

In this situation, if the default behavior is used, the user ID `MEDWAY\pieter` is resolved to the authorization ID `PIETER`. If the support for mapping a two-part user ID to a two-part authorization ID is enabled, the authorization ID would be `MEDWAY\PIETER`.

To enable DB2 to map two-part user IDs to two-part authorization IDs, DB2 supplies two sets of authentication plug-ins:

- One set exclusively maps a one-part user ID to a one-part authorization ID and maps a two-part user-ID to a one-part authorization ID.
- Another set maps both one-part user ID or two-part user ID to a two-part authorization ID.

If a user name in your work environment can be mapped to multiple accounts defined in different locations (such as local account, domain account, and trusted domain accounts), you can specify the plug-ins that enable two-part authorization ID mapping.

It is important to note that a one-part authorization ID, such as, `PIETER` and a two-part authorization ID that combines a domain and a user ID like `MEDWAY\pieter` are functionally distinct authorization IDs. The set of privileges associated with one of these authorization IDs can be completely distinct from the set of privileges associated with the other authorization ID. Care should be taken when working with one-part and two-part authorization IDs.

The following table identifies the kinds of plug-ins supplied by DB2 database systems, and the plug-in names for the specific authentication implementations.

Table 160. DB2 security plug-ins

Authentication type	Name of one-part user ID plug-in	Name of two-part user ID plug-in
User ID/password (client)	IBMOSauthclient	IBMOSauthclientTwoPart
User ID/password (server)	IBMOSauthserver	IBMOSauthserverTwoPart
Kerberos	IBMkrb5	IBMkrb5TwoPart

Note: On Windows 64-bit platforms, the characters ″64″ are appended to the plug-in names listed here.

When you specify an authentication type that requires a user ID/password or Kerberos plug-in, the plug-ins that are listed in the ″Name of one-part user ID plug-in″ column in the previous table are used by default.

To map a two-part user ID to a two-part authorization ID, you must specify that the two-part plug-in, which is not the default plug-in, be used. Security plug-ins are specified at the instance level by setting the security related database manager configuration parameters as follows:

For server authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBMOSauthserverTwoPart`
- `clnt_pw_plugin` to `IBMOSauthclientTwoPart`

For client authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_pw_plugin` to `IBMOSauthserverTwoPart`
- `clnt_pw_plugin` to `IBMOSauthclientTwoPart`

For Kerberos authentication that maps two-part user IDs to two-part authorization IDs, you must set:

- `srvcon_gssplugin_list` to `IBMOSkrb5TwoPart`
- `clnt_krb_plugin` to `IBMkrb5TwoPart`

The security plug-in libraries accept two-part user IDs specified in a Microsoft Windows Security Account Manager compatible format. For example, in the format: *domain\user ID*. Both the domain and user ID information will be used by the DB2 authentication and authorization processes at connection time.

You should consider implementing the two-part plug-ins when creating new databases to avoid conflicts with one-part authorization IDs in existing databases. New databases that use two-part authorization IDs must be created in a separate instance from databases that use single-part authorization IDs.

Related concepts:

- "DB2 and Windows security introduction" in *Administration Guide: Implementation*
- "Security plug-ins" on page 1397

Related tasks:

- "Authentication with groups and domain security (Windows)" in *Administration Guide: Implementation*

Related reference:

- "clnt_krb_plugin - Client Kerberos plug-in " on page 1467
- "clnt_pw_plugin - Client userid-password plug-in " on page 1467
- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server " on page 1469
- "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server " on page 1470

32-bit and 64-bit considerations for security plug-ins

In general, a 32-bit DB2 instance will use the 32-bit security plug-in and 64-bit DB2 instance will use the 64-bit security plug-in. However, on a 64-bit instance, DB2 supports 32-bit applications, which will require the 32-bit plug-in library.

A database instance where both the 32-bit and the 64-bit applications can run is known as a hybrid instance. If you have a hybrid instance and intend to run 32-bit applications, ensure that the required 32-bit security plug-ins are available in the 32-bit plug-in directory. For 64-bit DB2 instances on Linux and UNIX-based operating systems, excluding Linux on IPF, the directories `security32` and `security64` appear. For a 64-bit DB2 instance on Windows on X64 or IPF, both 32-bit and 64-bit security plug-ins are located in the same directory, but 64-bit plug-in names have a suffix, "64".

If you want to migrate from a 32-bit instance to a 64-bit instance, you should obtain versions of your security plug-ins that are recompiled for 64-bit.

32-bit and 64-bit considerations for security plug-ins

If you acquired your security plug-ins from a vendor that does not supply 64-bit plug-in libraries, you can implement a 64-bit stub that executes a 32-bit application. In this situation, the security plug-in is an external program rather than a library.

Related concepts:

- "Security plug-in library locations" on page 1401
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397

Related tasks:

- "Migrating 32-bit database applications to run on 64-bit instances" in *Migration Guide*

Security plug-in problem determination

Problems with security plug-ins are reported in two ways: through SQL errors and through the administration notification log.

Following are the SQLCODE values related to security plug-ins:

- SQLCODE -1365 is returned when a plug-in error occurs during `db2start` or `db2stop`.
- SQLCODE -1366 is returned whenever there is a local authorization problem.
- SQLCODE -30082 is returned for all connection-related plug-in errors.

The administration notification log is a good resource for debugging and administrating security plug-ins. To see the administration notification log on UNIX, check `sqllib/db2dump/`*`instance name`*`.nfy`. To see the administration notification log on Windows operating systems, use the Event Viewer tool. The Event Viewer tool can be found by navigating from the Windows operating system "Start" button to `Settings -> Control Panel -> Administrative Tools -> Event Viewer`. Following are the administration notification log values related to security plug-ins:

- 13000 indicates that a call to a GSS-API security plug-in API failed with an error, and returned an optional error message.

  ```
  SQLT_ADMIN_GSS_API_ERROR (13000)
  Plug-in "plug-in name" received error code "error code" from
  GSS API  "gss api name" with the error message "error message"
  ```
- 13001 indicates that a call to a DB2 security plug-in API failed with an error, and returned an optional error message.

  ```
  SQLT_ADMIN_PLUGIN_API_ERROR(13001)
  Plug-in "plug-in name" received error code "error code" from DB2
  security plug-in API "gss api name" with the error message
  "error message"
  ```
- 13002 indicates that DB2 failed to unload a plug-in.

  ```
  SQLT_ADMIN_PLUGIN_UNLOAD_ERROR (13002)
  Unable to unload plug-in "plug-in name". No further action required.
  ```
- 13003 indicates a bad principal name.

  ```
  SQLT_ADMIN_INVALID_PRIN_NAME (13003)
  The principal name "principal name" used for "plug-in name"
  is invalid. Fix the principal name.
  ```
- 13004 indicates that the plug-in name is not valid. Path separators (On UNIX "/" and on Windows "\") are not allowed in the plug-in name.

```
SQLT_ADMIN_INVALID_PLGN_NAME (13004)
The plug-in name "plug-in name" is invalid. Fix the plug-in name.
```

- 13005 indicates that the security plug-in failed to load. Ensure the plug-in is in the correct directory and that the appropriate database manager configuration parameters are updated.

```
SQLT_ADMIN_PLUGIN_LOAD_ERROR (13005)
Unable to load plug-in "plug-in name". Verify the plug-in existence and
directory where it is located is correct.
```

- 13006 indicates that an unexpected error was encountered by a security plug-in. Gather all the `db2support` information, if possible capture a `db2trc`, and then call IBM support for further assistance.

```
SQLT_ADMIN_PLUGIN_UNEXP_ERROR (13006)
Plug-in encountered unexpected error. Contact IBM Support for further assistance.
```

Note: If you are using security plug-ins on a Windows 64-bit database server and are seeing a load error for a security plug-in, see the topics ″32-bit and 64-bit considerations for security plug-ins″ and ″Security plug-in naming conventions″. The 64-bit plug-in library requires the suffix ″64″ on the library name, but the entry in the security plug-in database manager configuration parameters should not indicate this suffix.

Related concepts:

- "32-bit and 64-bit considerations for security plug-ins" on page 1405
- "Error information in the SQLCODE, SQLSTATE, and SQLWARN fields" in *Developing Embedded SQL Applications*
- "Security plug-in APIs" on page 1425
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397
- "SQLCODE and SQLSTATE Differences among IBM Relational Database Systems" in *Developing SQL and External Routines*

Related reference:

- "db2trc - Trace command" in *Command Reference*
- "APIs for group retrieval plug-ins" on page 1427
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Required APIs and definitions for GSS-API authentication plug-ins" on page 1460

Deploying a group retrieval plug-in

To customize the DB2 security system's group retrieval behavior, you can develop your own group retrieval plug-in or buy one from a third party.

After you acquire a group retrieval plug-in that is suitable for your database management system, you can deploy it.

Procedure:

To deploy a group retrieval plug-in on the database server, perform the following steps:

1. Copy the group retrieval plug-in library into the server's group plug-in directory.

2. Update the database manager configuration parameter *group_plugin* with the name of the plug-in.

To deploy a group retrieval plug-in on database clients, perform the following steps:
1. Copy the group retrieval plug-in library in the client's group plug-in directory.
2. On the database client, update the database manager configuration parameter *group_plugin* with the name of the plug-in.

Related concepts:
- "Security plug-in library locations" on page 1401
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397

Related tasks:
- "Deploying a GSS-API plug-in" on page 1410
- "Deploying a Kerberos plug-in" on page 1411
- "Deploying a user ID/password plug-in" on page 1408

Related reference:
- "group_plugin - Group plug-in " on page 1468
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Deploying a user ID/password plug-in

To customize the DB2 security system's user ID/password authentication behavior, you can develop your own user ID/password authentication plug-ins or buy one from a third party.

After you acquire user ID/password authentication plug-ins that are suitable for your database management system, you can deploy them.

Depending on their intended usage, all user ID-password based authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used both for local authorization checking and for validating the client when it attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authorization ID exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

In most situations, user ID/password authentication requires only a server-side plug-in. It is possible, though generally deemed less useful, to have only a client user ID/password plug-in. It is possible, though quite unusual to require matching user ID/password plug-ins on both the client and the server.

Procedure:

To deploy a user ID/password authentication plug-in on the database server, perform the following steps on the database server:
1. Copy the user ID/password authentication plug-in library in the server plug-in directory.

2. Update the database manager configuration parameter *srvcon_pw_plugin* with the name of the server plug-in.

   This plug-in is used by the server when it is handling CONNECT and ATTACH requests.
3. Either:
   - Set the database manager configuration parameter *srvcon_auth* to the CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP authentication type. Or:
   - Set the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and set *authentication* to CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP authentication type.

To deploy a user ID/password authentication plug-in on database clients, perform the following steps on each client:

1. Copy the user ID/password authentication plug-in library in the client plug-in directory.
2. Update the database manager configuration parameter *clnt_pw_plugin* with the name of the client plug-in.

   This plug-in is loaded and called regardless of where the authentication is being done, not only when the database configuration parameter, *authentication* is set to CLIENT.

For local authorization on a client, server, or gateway using a user ID/password authentication plug-in, perform the following steps on each client, server, or gateway:

1. Copy the user ID/password authentication plug-in library in the client plug-in directory on the client, server, or gateway.
2. Update the database manager configuration parameter *clnt_pw_plugin* with the name of the plug-in.
3. Set the *authentication* database manager configuration parameter to CLIENT, SERVER, SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP.

Related concepts:

- "Security plug-in library locations" on page 1401
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397

Related tasks:

- "Deploying a group retrieval plug-in" on page 1407
- "Deploying a GSS-API plug-in" on page 1410
- "Deploying a Kerberos plug-in" on page 1411

Related reference:

- "authentication - Authentication type " on page 1497
- "clnt_pw_plugin - Client userid-password plug-in " on page 1467
- "srvcon_auth - Authentication type for incoming connections at the server " on page 1469
- "srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server " on page 1470
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Deploying a GSS-API plug-in

To customize the DB2 security system's authentication behavior, you can develop your own authentication plug-ins using the GSS-API, or buy one from a third party.

After you acquire GSS-API authentication plug-ins that are suitable for your database management system, you can deploy them.

In the case of any plug-ins types other than Kerberos, you must have matching plug-in names on the client and the server along with the same plug-in type. The plug-ins on the client and server need not be from the same vendor, but they must generate and consume compatible GSS-API tokens. Any combination of Kerberos plug-ins deployed on the client and the server is acceptable since Kerberos plug-ins are standardized. However, different implementations of less standardized GSS-API mechanisms, such as *x.509* certificates, might only be partially compatible with DB2 database systems.

Depending on their intended usage, all GSS-API authentication plug-ins must be placed in either the client plug-in directory or the server plug-in directory. If a plug-in is placed in the client plug-in directory, it will be used for local authorization checking and when a client attempts to connect with the server. If the plug-in is placed in the server plug-in directory, it will be used for handling incoming connections to the server and for checking whether an authorization ID exists and is valid whenever the GRANT statement is issued without specifying either the keyword USER or GROUP.

Procedure:

To deploy a GSS-API authentication plug-in on the database server, perform the following steps on the server:

1. Copy the GSS-API authentication plug-in library in the server plug-in directory. You can copy numerous GSS-API plug-ins into this directory.
2. Update the database manager configuration parameter *srvcon_gssplugin_list* with an ordered, comma-delimited list of the names of the plug-ins installed in the GSS-API plug-in directory.
3. Either:
   - Setting the database manager configuration parameter *srvcon_auth* to GSSPLUGIN or GSS_SERVER_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method. Or:
   - Setting the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and setting *authentication* to GSSPLUGIN or GSS_SERVER_ENCRYPT is a way to enable the server to use GSSAPI PLUGIN authentication method.

To deploy a GSS-API authentication plug-in on database clients, perform the following steps on each client:

1. Copy the GSS-API authentication plug-in library in the client plug-in directory. You can copy numerous GSS-API plug-ins into this directory. The client selects a GSS-API plug-in for authentication during a CONNECT or ATTACH operation by picking the first GSS-API plug-in contained in the server's plug-in list that is available on the client.

2. Optional: Catalog the databases that the client will access, indicating that the client will only accept a GSS-API authentication plug-in as the authentication mechanism. For example:

   ```
   CATALOG DB testdb AT NODE testnode AUTHENTICATION GSSPLUGIN
   ```

For local authorization on a client, server, or gateway using a GSS-API authentication plug-in, perform the following steps:

1. Copy the GSS-API authentication plug-in library in the client plug-in directory on the client, server, or gateway.
2. Update the database manager configuration parameter *local_gssplugin* with the name of the plug-in.
3. Set the *authentication* database manager configuration parameter to GSSPLUGIN, or GSS_SERVER_ENCRYPT.

Related concepts:

- "Security plug-in library locations" on page 1401
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397

Related tasks:

- "Deploying a group retrieval plug-in" on page 1407
- "Deploying a Kerberos plug-in" on page 1411
- "Deploying a user ID/password plug-in" on page 1408

Related reference:

- "srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server " on page 1469
- "authentication - Authentication type " on page 1497
- "local_gssplugin - GSS API plug-in used for local instance level authorization " on page 1468
- "srvcon_auth - Authentication type for incoming connections at the server " on page 1469
- "CATALOG DATABASE " on page 458
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Deploying a Kerberos plug-in

To customize the DB2 security system's Kerberos authentication behavior, you can develop your own Kerberos authentication plug-ins or buy one from a third party.

After you acquire Kerberos authentication plug-ins that are suitable for your database management system, you can deploy them.

Note that the Kerberos security plug-in will not support IPv6.

Procedure:

To deploy a Kerberos authentication plug-in on the database server, perform the following steps on the server:

1. Copy the Kerberos authentication plug-in library in the server plug-in directory.

2. Update the database manager configuration parameter *srvcon_gssplugin_list*, which is presented as an ordered, comma delimited list, to include the Kerberos server plug-in name. Only one plug-in in this list can be a Kerberos plug-in.

   If this list is blank and *authentication* is set to KERBEROS or KRB_SVR_ENCRYPT, the default DB2 Kerberos plug-in: IBMkrb5 will be used.
3. Either:
   - Set the database manager configuration parameter *srvcon_auth* to the KERBEROS or KRB_SERVER_ENCRYPT authentication type. (You can deploy a KERBEROS plugin and still use GSSPLUGIN or GSS_SERVER_ENCRYPT) Or:
   - Set the database manager configuration parameter *srvcon_auth* to NOT_SPECIFIED and set *authentication* to KERBEROS or KRB_SERVER_ENCRYPT authentication type.

To deploy a Kerberos authentication plug-in on database clients, perform the following steps on each client:
1. Copy the Kerberos authentication plug-in library in the client plug-in directory.
2. Update the database manager configuration parameter *clnt_krb_plugin* with the name of the Kerberos plug-in.

   If *clnt_krb_plugin* is blank, DB2 assumes that the client cannot use Kerberos authentication. This setting is only appropriate when the server cannot support plug-ins. If both the server and the client support security plug-ins, the default server plug-in, *IBMkrb5* would be used over the client value of *clnt_krb_plugin*.

   For local authorization on a client, server, or gateway using a Kerberos authentication plug-in, perform the following steps:
   a. Copy the Kerberos authentication plug-in library in the client plug-in directory on the client, server, or gateway.
   b. Update the database manager configuration parameter *clnt_krb_plugin* with the name of the plug-in.
   c. Set the *authentication* database manager configuration parameter to KERBEROS, or KRB_SERVER_ENCRYPT.
3. Optional: Catalog the databases that the client will access, indicating that the client will only use a Kerberos authentication plug-in. For example:

```
CATALOG DB testdb AT NODE testnode AUTHENTICATION KERBEROS
       TARGET PRINCIPAL service/host@REALM
```

Note: For platforms supporting Kerberos, the `IBMkrb5` library will be present in the client plug-in directory. DB2 will recognize this library as a valid GSS-API plug-in, because Kerberos plug-ins are implemented using GSS-API plug-in.

Related concepts:
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397
- "Security plug-in library locations" on page 1401

Related tasks:
- "Deploying a group retrieval plug-in" on page 1407
- "Deploying a GSS-API plug-in" on page 1410
- "Deploying a user ID/password plug-in" on page 1408

Related reference:

- “authentication - Authentication type ” on page 1497
- “clnt_krb_plugin - Client Kerberos plug-in ” on page 1467
- “srvcon_auth - Authentication type for incoming connections at the server ” on page 1469
- “srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server ” on page 1469
- “CATALOG DATABASE ” on page 458
- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629

Restrictions on security plug-ins

The following are restrictions on the use of security plug-ins:

DB2 database family support restrictions:

You cannot use a GSS-API plug-in to authenticate connections between DB2 clients on Linux, UNIX, and Windows and another DB2 family servers such as DB2 for z/OS. You also cannot authenticate connections from another DB2 database family product, acting as a client, to a DB2 server on Linux, UNIX, or Windows.

If you use a DB2 client on Linux, UNIX, or Windows to connect to other DB2 database family servers, you can use client-side user ID/password plug-ins (such as the IBM-shipped operating system authentication plug-in), or you can write your own user ID/password plug-in. You can also use the built-in Kerberos plug-ins, or implement your own.

With a DB2 client on Linux, UNIX, or Windows, you should not catalog a database using the GSSPLUGIN authentication type.

WebSphere Federation Server support restrictions:

DB2 II does not support the use of delegated credentials from a GSS_API plug-in to establish outbound connections to data sources. Connections to data sources must continue to use the CREATE USER MAPPING command.

Database Administration Server support restrictions:

The DB2 Administration Server (DAS) does not support security plug-ins. The DAS only supports the operating system authentication mechanism.

Security plug-in problem and restriction for DB2 clients (Windows):

When developing security plug-ins that will be deployed in DB2 clients on Windows operating systems, do not unload any auxiliary libraries in the plug-in termination function. This restriction applies to all types of client security plug-ins, including group, user ID and password, Kerberos, and GSS-API plug-ins. Since these termination APIs such as db2secPluginTerm, db2secClientAuthPluginTerm and db2secServerAuthPluginTerm are not called on any Windows platform, you need to do the appropriate resource cleanup.

This restriction is related to cleanup issues associated with the unloading of DLLs on Windows.

Loading plug-in libraries on AIX with extension of .a or .so:

On AIX, security plug-in libraries can have a file name extension of `.a` or `.so`. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of `.a`

  Plug-in libraries with file name extensions of .a are assumed to be archives containing shared object members. These members must be named `shr.o` (32-bit) or `shr64.o` (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

  For example, to build a 32-bit archive style plug-in library:

  ```
  xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
  ar rv MyPlugin.a shr.o
  ```
- Plug-in libraries with a file name extension of `.so`

  Plug-in libraries with file name extensions of .so are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

  ```
  xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
  ```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

GSS-API security plug-ins do not support message encryption and signing:

Message encryption and signing is not available in GSS-API security plug-ins.

Related reference:
- "db2secClientAuthPluginTerm - Clean up client authentication plug-in resources" on page 1443
- "db2secPluginTerm - Clean up group plug-in resources" on page 1430
- "db2secServerAuthPluginTerm - Clean up server authentication plug-in resources" on page 1456

Chapter 45. Developing security plug-ins

This chapter will focus on creating user specific security plug-ins along with restrictions and error messages. To develop a security plug-in, you need an initialization function, and the plug-in must return a integer value to indicate success or failure of the execution of the API.

How DB2 loads security plug-ins

Each plug-in library must contain an initialization function with a specific name determined by the plug-in type:

- Server side authentication plug-in: `db2secServerAuthPluginInit()`
- Client side authentication plug-in: `db2secClientAuthPluginInit()`
- Group plug-in: `db2secGroupPluginInit()`

This function is known as the plug-in initialization function. The plug-in initialization function initializes the specified plug-in and provides DB2 with information that it requires to call the plug-in's functions. The plug-in initialization function accepts the following parameters:

- The highest version number of the function pointer structure that the DB2 instance invoking the plugin can support
- A pointer to a structure containing pointers to all the APIs requiring implementation
- A pointer to a function that adds log messages to the db2diag.log file
- A pointer to an error message string
- The length of the error message

The following is a function signature for the initialization function of a group retrieval plug-in:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit(
  db2int32 version,
  void *group_fns,
  db2secLogMessage *logMessage_fn,
  char **errormsg,
  db2int32  *errormsglen);
```

Note: If the plug-in library is compiled as C++, all functions must be declared with: `extern "C"`. DB2 relies on the underlying operating system dynamic loader to handle the C++ constructors and destructors used inside of a C++ user-written plug-in library.

The initialization function is the only function in the plug-in library that uses a prescribed function name. The other plug-in functions are referenced through function pointers returned from the initialization function. Server plug-ins are loaded when the DB2 server starts. Client plug-ins are loaded when required on the client. Immediately after DB2 loads a plug-in library, it will resolve the location of this initialization function and call it. The specific task of this function is as follows:

- Cast the functions pointer to a pointer to an appropriate functions structure
- Fill in the pointers to the other functions in the library
- Fill in the version number of the function pointer structure being returned

DB2 can potentially call the plug-in initialization function more than once. This situation can occur when an application dynamically loads the DB2 client library, unloads it, and reloads it again, then performs authentication functions from a plug-in both before and after reloading. In this situation, the plug-in library might not be unloaded and then re-loaded; however, this behavior varies depending on the operating system.

Another example of DB2 issuing multiple calls to a plug-in initialization function occurs during the execution of stored procedures or federated system calls, where the database server can itself act as a client. If the client and server plug-ins on the database server are in the same file, DB2 could call the plug-in initialization function twice.

If the plug-in detects that `db2secGroupPluginInit` is called more than once, it should handle this event as if it was directed to terminate and reinitialize the plug-in library. As such, the plug-in initialization function should do the entire cleanup tasks that a call to `db2secPluginTerm` would do before returning the set of function pointers again.

On a DB2 server running on a UNIX or Linux-based operating system, DB2 can potentially load and initialize plug-in libraries more than once in different processes.

Related concepts:

- "Security plug-ins" on page 1397

Related reference:

- "Calling sequences for the security plug-in APIs" on page 1416
- "db2secClientAuthPluginInit - Initialize client authentication plug-in" on page 1441
- "db2secGroupPluginInit - Initialize group plug-in" on page 1428
- "db2secPluginTerm - Clean up group plug-in resources" on page 1430
- "db2secServerAuthPluginInit - Initialize server authentication plug-in" on page 1454
- "Restrictions for developing security plug-in libraries" on page 1419

Calling sequences for the security plug-in APIs

There are five main scenarios in which the DB2 database manager will call security plug-in APIs:

- On a client for a database connection (implicit and explicit)
  - CLIENT
  - Server based (SERVER, SERVER_ENCRYPT, DATA_ENCRYPT)
  - GSSAPI and Kerberos
- On a client, server, or gateway for local authorization

- On a server for a database connection
- On a server for a grant statement
- On a server to get a list of groups to which an authorization ID belongs

Note: The DB2 database servers treat database actions requiring local authorizations, such as `db2start`, `db2stop`, and `db2trc` like client applications.

For each of these operations, the sequence with which the DB2 database manager calls the security plug-in APIs is different. Following are the sequences of APIs called by the DB2 database manager for each of these scenarios.

CLIENT - implicit
When the user-configured authentication type is CLIENT, the DB2 client application will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secValidatePassword();`
- `db2secFreetoken();`

For an implicit authentication, that is, when you connect without specifying a particular user ID or password, the `db2secValidatePassword` API is called if you are using a user ID/password plug-in. This API permits plug-in developers to prohibit implicit authentication if necessary.

CLIENT - explicit
On an explicit authentication, that is, when you connect to a database in which both the user ID and password are specified, if the *authentication* database manager configuration parameter is set to CLIENT the DB2 client application will call the following security plug-in APIs multiple times if the implementation requires it:

- `db2secRemapUserid();`
- `db2secValidatePassword();`
- `db2secFreeToken();`

Server based (SERVER, SERVER_ENCRYPT, DATA_ENCRYPT) - implicit
On an implicit authentication, when the client and server have negotiated user ID/password authentication (for instance, when the *srvcon_auth* parameter at the server is set to SERVER; SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP), the client application will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secFreeToken();`

Server based (SERVER, SERVER_ENCRYPT, DATA_ENCRYPT) - explicit
On an explicit authentication, when the client and server have negotiated userid/password authentication (for instance, when the srvcon_auth parameter at the server is set to SERVER; SERVER_ENCRYPT, DATA_ENCRYPT, or DATA_ENCRYPT_CMP), the client application will call the following security plug-in APIs:

- `db2secRemapUserid();`

GSSAPI and Kerberos - implicit
On an implicit authentication, when the client and server have negotiated GSS-API or Kerberos authentication (for instance, when the *srvcon_auth* parameter at the server is set to KERBEROS; KRB_SERVER_ENCRYPT, GSSPLUGIN, or GSS_SERVER_ENCRYPT), the client application will call

the following security plug-in APIs. (The call to gss_init_sec_context() will use GSS_C_NO_CREDENTIAL as the input credential.)

- `db2secGetDefaultLoginContext();`
- `db2secProcessServerPrincipalName();`
- `gss_init_sec_context();`
- `gss_release_buffer();`
- `gss_release_name();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

With multi-flow GSS-API support, `gss_init_sec_context()` can be called multiple times if the implementation requires it.

GSSAPI and Kerberos - explicit

If the negotiated authentication type is GSS-API or Kerberos, the client application will call the following security plug-in APIs for GSS-API plug-ins in the following sequence. These APIs are used for both implicit and explicit authentication unless otherwise stated.

- `db2secProcessServerPrincipalName();`
- `db2secGenerateInitialCred();` (For explicit authentication only)
- `gss_init_sec_context();`
- `gss_release_buffer ();`
- `gss_release_name();`
- `gss_release_cred();`
- `db2secFreeInitInfo();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`

The API `gss_init_sec_context()` may be called multiple times if a mutual authentication token is returned from the server and the implementation requires it.

On a client, server, or gateway for local authorization

For a local authorization, the DB2 command being used will call the following security plug-in APIs:

- `db2secGetDefaultLoginContext();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

These APIs will be called for both user ID/password and GSS-API authentication mechanisms.

On a server for a database connection

For a database connection on the database server, the DB2 agent process or thread will call the following security plug-in APIs for the user ID/password authentication mechanism:

- `db2secValidatePassword();` Only if the *authentication* database configuration parameter is not CLIENT
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

For a CONNECT to a database, the DB2 agent process or thread will call the following security plug-in APIs for the GSS-API authentication mechanism:

- `gss_accept_sec_context();`
- `gss_release_buffer();`
- `db2secGetAuthIDs();`
- `db2secGetGroupsForUser();`
- `gss_delete_sec_context();`
- `db2secFreeToken();`
- `db2secFreeGroupList();`

On a server for a GRANT statement

For a GRANT statement that does not specify the USER or GROUP keyword, (for example, ″`GRANT CONNECT ON DATABASE TO user1`″), the DB2 agent process or thread must be able to determine if `user1` is a user, a group, or both. Therefore, the DB2 agent process or thread will call the following security plug-in APIs:

- `db2secDoesGroupExist();`
- `db2secDoesAuthIDExist();`

On a server to get a list of groups to which an authid belongs

From your database server, when you need to get a list of groups to which an authorization ID belongs, the DB2 agent process or thread will call the following security plug-in API with only the authorization ID as input:

- `db2secGetGroupsForUser();`

There will be no token from other security plug-ins.

Related concepts:

- "How DB2 loads security plug-ins" on page 1415
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Restrictions for developing security plug-in libraries

Following are restrictions for developing plug-in libraries.

C-linkage

Plug-in libraries must be linked with C-linkage. Header files providing the prototypes, data structures needed to implement the plug-ins, and error code definitions are provided for C/C++ only. Functions that DB2 will resolve at load time must be declared with extern ″C″ if the plug-in library is compiled as C++.

.NET common language runtime is not supported

The .NET common language runtime (CLR) is not supported for compiling and linking source code for plug-in libraries.

Signal handlers

Plug-in libraries must not install signal handlers or change the signal mask, because this will interfere with DB2's signal handlers. Interfering with the DB2 signal handlers could seriously interfere with DB2's ability to report and recover from errors, including traps in the plug-in code itself. Plug-in libraries should also never throw C++ exceptions, as this can also interfere with DB2's error handling.

Restrictions for developing security plug-in libraries

Thread-safe
: Plug-in libraries must be thread-safe and re-entrant. The plug-in initialization function is the only API that is not required to be re-entrant. The plug-in initialization function could potentially be called multiple times from different processes; in which case, the plug-in will cleanup all used resources and reinitialize itself.

Exit handlers and overriding standard C library and operating system calls
: Plug-in libraries should not override standard C library or operating system calls. Plug-in libraries should also not install exit handlers or `pthread_atfork` handlers. The use of exit handlers is not recommended because they could be unloaded before the program exits.

Library dependencies
: On Linux or UNIX, the processes that load the plug-in libraries can be `setuid` or `setgid`, which means that they will not be able to rely on the `$LD_LIBRARY_PATH`, `$SHLIB_PATH`, or `$LIBPATH` environment variables to find dependent libraries. Therefore, plug-in libraries should not depend on additional libraries, unless any dependant libraries are accessible through other methods, such as the following:

- By being in `/lib` or `/usr/lib`
- By having the directories they reside in being specified OS-wide (such as in the `ld.so.conf` file on Linux)
- By being specified in the `RPATH` in the plug-in library itself

This restriction is not applicable to Windows operating systems.

Symbol collisions
: When possible, plug-in libraries should be compiled and linked with any available options that reduce the likelihood of symbol collisions, such as those that reduce unbound external symbolic references. For example, use of the ″-Bsymbolic″ linker option on HP, Sun Solaris, and Linux can help prevent problems related to symbol collisions. However, for plug-ins written on AIX, do not use the `"-brtl"` linker option explicitly or implicitly.

32-bit and 64-bit applications
: 32-bit applications must use 32-bit plug-ins. 64-bit applications must use 64-bit plug-ins. Refer to the topic 32-bit and 64-bit considerations for security plug-ins for more details.

Text strings
: Input text strings are not guaranteed to be null-terminated, and output strings are not required to be null-terminated. Instead, integer lengths are given for all input strings, and pointers to integers are given for lengths to be returned.

Passing authorization ID parameters
: An authorization ID (authid) parameter that DB2 passes into a plug-in (an input authid parameter) will contain an upper-case authid, with padded blanks removed. An authid parameter that a plug-in returns to DB2 (an output authid parameter) does not require any special treatment, but DB2 will fold the authid to upper-case and pad it with blanks according to the internal DB2 standard.

Size limits for parameters
: The plug-in APIs use the following as length limits for parameters:

```
#define DB2SEC_MAX_AUTHID_LENGTH 255
#define DB2SEC_MAX_USERID_LENGTH 255
#define DB2SEC_MAX_USERNAMESPACE_LENGTH 255
#define DB2SEC_MAX_PASSWORD_LENGTH 255
#define DB2SEC_MAX_DBNAME_LENGTH 128
```

Note: The maximum AUTHID length accepted by DB2 database systems in v9 is 30 characters.

A particular plug-in implementation may require or enforce smaller maximum lengths for the authorization IDs, user IDs, and passwords. In particular, the operating system authentication plug-ins supplied with DB2 database systems are restricted to the maximum user, group and namespace length limits enforced by the operating system for cases where the operating system limits are lower than those stated above.

Security plug-in library extensions in AIX

On AIX systems, security plug-in libraries can have a file name extension of *.a* or *.so*. The mechanism used to load the plug-in library depends on which extension is used:

- Plug-in libraries with a file name extension of *.a* are assumed to be archives containing shared object members. These members must be named *shr.o* (32-bit) or *shr64.o* (64-bit). A single archive can contain both the 32-bit and 64-bit members, allowing it to be deployed on both types of platforms.

  For example, to build a 32-bit archive style plug-in library:

  ```
  xlc_r -qmkshrobj -o shr.o MyPlugin.c -bE:MyPlugin.exp
  ar rv MyPlugin.a shr.o
  ```
- Plug-in libraries with a file name extension of *.so* are assumed to be dynamically loadable shared objects. Such an object is either 32-bit or 64-bit, depending on the compiler and linker options used when it was built. For example, to build a 32-bit plug-in library:

  ```
  xlc_r -qmkshrobj -o MyPlugin.so MyPlugin.c -bE:MyPlugin.exp
  ```

On all platforms other than AIX, security plug-in libraries are always assumed to be dynamically loadable shared objects.

Related concepts:

- "32-bit and 64-bit considerations for security plug-ins" on page 1405
- "How DB2 loads security plug-ins" on page 1415
- "Security plug-in library locations" on page 1401
- "Security plug-in naming conventions" on page 1402
- "Security plug-ins" on page 1397

Return codes for security plug-ins

All security plug-in APIs must return an integer value to indicate the success or failure of the execution of the API. A return code value of 0 indicates that the API ran successfully. All negative return codes, with the exception of -3, -4, and -5, indicate that the API encountered an error.

All negative return codes returned from the security-plug-in APIs are mapped to SQLCODE -1365, SQLCODE -1366, or SQLCODE -30082, with the exception of return codes with the -3, -4, or -5. The values -3, -4, and -5 are used to indicate whether or not an AUTHORIZATION ID represents a valid user or group.

Return codes for security plug-ins

All the security plug-in API return codes are defined in db2secPlugin.h, which can be found in the DB2 include directory: SQLLIB/include.

Details regarding all of the security plug-in return codes are presented in the following table:

Table 161. Security plug-in return codes

Return code	Define value	Meaning	Applicable APIs
0	`DB2SEC_PLUGIN_OK`	The plug-in API executed successfully.	All
-1	`DB2SEC_PLUGIN_UNKNOWNERROR`	The plug-in API encountered an unexpected error.	All
-2	`DB2SEC_PLUGIN_BADUSER`	The user ID passed in as input is not defined.	`db2secGenerateInitialCred` `db2secValidatePassword` `db2secRemapUserid` `db2secGetGroupsForUser`
-3	`DB2SEC_PLUGIN _INVALIDUSERORGROUP`	No such user or group.	`db2secDoesAuthIDExist` `db2secDoesGroupExist`
-4	`DB2SEC_PLUGIN _USERSTATUSNOTKNOWN`	Unknown user status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group.	`db2secDoesAuthIDExist`
-5	`DB2SEC_PLUGIN _GROUPSTATUSNOTKNOWN`	Unknown group status. This is not treated as an error by DB2; it is used by a GRANT statement to determine if an authid represents a user or an operating system group.	`db2secDoesGroupExist`
-6	`DB2SEC_PLUGIN_UID_EXPIRED`	User ID expired.	`db2secValidatePassword` `db2GetGroupsForUser` `db2secGenerateInitialCred`
-7	`DB2SEC_PLUGIN_PWD_EXPIRED`	Password expired.	`db2secValidatePassword` `db2GetGroupsForUser` `db2secGenerateInitialCred`
-8	`DB2SEC_PLUGIN_USER_REVOKED`	User revoked.	`db2secValidatePassword` `db2GetGroupsForUser`
-9	`DB2SEC_PLUGIN _USER_SUSPENDED`	User suspended.	`db2secValidatePassword` `db2GetGroupsForUser`
-10	`DB2SEC_PLUGIN_BADPWD`	Bad password.	`db2secValidatePassword` `db2secRemapUserid` `db2secGenerateInitialCred`
-11	`DB2SEC_PLUGIN _BAD_NEWPASSWORD`	Bad new password.	`db2secValidatePassword` `db2secRemapUserid`
-12	`DB2SEC_PLUGIN _CHANGEPASSWORD _NOTSUPPORTED`	Change password not supported.	`db2secValidatePassword` `db2secRemapUserid` `db2secGenerateInitialCred`
-13	`DB2SEC_PLUGIN_NOMEM`	Plug-in attempt to allocate memory failed due to insufficient memory.	All
-14	`DB2SEC_PLUGIN_DISKERROR`	Plug-in encountered a disk error.	All
-15	`DB2SEC_PLUGIN_NOPERM`	Plug-in attempt to access a file failed because of wrong permissions on the file.	All
-16	`DB2SEC_PLUGIN_NETWORKERROR`	Plug-in encountered a network error.	All
-17	`DB2SEC_PLUGIN _CANTLOADLIBRARY`	Plug-in is unable to load a required library.	`db2secGroupPluginInit` `db2secClientAuthPluginInit` `db2secServerAuthPluginInit`

Table 161. Security plug-in return codes (continued)

Return code	Define value	Meaning	Applicable APIs
-18	DB2SEC_PLUGIN_CANT _OPEN_FILE	Plug-in is unable to open and read a file for a reason other than missing file or inadequate file permissions.	All
-19	DB2SEC_PLUGIN_FILENOTFOUND	Plug-in is unable to open and read a file, because the file is missing from the file system.	All
-20	DB2SEC_PLUGIN _CONNECTION_DISALLOWED	The plug-in is refusing the connection because of the restriction on which database is allowed to connect, or the TCP/IP address cannot connect to a specific database.	All server-side plug-in APIs.
-21	DB2SEC_PLUGIN_NO_CRED	GSS API plug-in only: initial client credential is missing.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-22	DB2SEC_PLUGIN_CRED_EXPIRED	GSS API plug-in only: client credential has expired.	db2secGetDefaultLoginContext db2secServerAuthPluginInit
-23	DB2SEC_PLUGIN _BAD_PRINCIPAL_NAME	GSS API plug-in only: the principal name is invalid.	db2secProcessServerPrincipalName
-24	DB2SEC_PLUGIN _NO_CON_DETAILS	This return code is returned by the `db2secGetConDetails` callback (for example, from DB2 to the plug-in) to indicate that DB2 is unable to determine the client's TCP/IP address.	db2secGetConDetails
-25	DB2SEC_PLUGIN _BAD_INPUT_PARAMETERS	Some parameters are not valid or are missing when plug-in API is called.	All
-26	DB2SEC_PLUGIN _INCOMPATIBLE_VER	The version of the APIs reported by the plug-in is not compatible with DB2.	db2secGroupPluginInit db2secClientAuthPluginInit db2secServerAuthPluginInit
-27	DB2SEC_PLUGIN_PROCESS_LIMIT	Insufficient resources are available for the plug-in to create a new process.	All
-28	DB2SEC_PLUGIN_NO_LICENSES	The plug-in encountered a user license problem. A possibility exists that the underlying mechanism license has reached the limit.	All

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-in problem determination" on page 1406
- "Security plug-ins" on page 1397

Error message handling for security plug-ins

When an error occurs in a security plug-in API, the API can return an ASCII text string in the `errormsg` field to provide a more specific description of the problem than the return code. For instance, the `errormsg` string can contain `"File /home/db2inst1/mypasswd.txt does not exist."` DB2 will write this entire string into the DB2 administration notification log, and will also include a truncated version as a token in some SQL messages. Because tokens in SQL messages can only be of limited length, these messages should be kept short, and important variable portions of these messages should appear at the front of the string. To aid in debugging, consider adding the name of the security plug-in to the error message.

For non-urgent errors, such as password expired errors, the `errormsg` string will only be dumped when the DIAGLEVEL database manager configuration parameter is set at 4.

The memory for these error messages must be allocated by the security plug-in. Therefore, the plug-ins must also provide an API to free this memory: `db2secFreeErrormsg`.

The `errormsg` field will only be checked by DB2 if an API returns a non-zero value. Therefore, the plug-in should not allocate memory for this returned error message if there is no error.

At initialization time a message logging function pointer, `logMessage_fn`, is passed to the group, client, and server plug-ins. The plug-ins can use the function to log any debugging information to db2diag.log. For example:

```
// Log an message indicate init successful
(*(logMessage_fn))(DB2SEC_LOG_CRITICAL,
                   "db2secGroupPluginInit successful",
                    strlen("db2secGroupPluginInit successful"));
```

For more details about each parameter for the `db2secLogMessage` function, refer to the initialization API for each of the plug-in types.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-in problem determination" on page 1406
- "Security plug-ins" on page 1397
- "CLI trace files" in *Troubleshooting Guide*

Related reference:

- "Return codes for security plug-ins" on page 1421

Chapter 46. Security plug-in APIs

Security plug-in APIs

To enable you to customize the DB2 database system authentication and group membership lookup behavior, DB2 provides APIs that you can use to modify existing plug-ins or build new security plug-ins.

When you develop a security plug-in, you need to implement the standard authentication or group membership lookup functions that DB2 will invoke. For the three available types of plug-ins, the functionality you need to implement is as follows:

Group retrieval
: Retrieves group membership information for a given user and determines if a given string represents a valid group name.

User ID/password authentication
: Authentication that identifies the default security context (client only), validates and optionally changes a password, determines if a given string represents a valid user (server only), modifies the user ID or password provided on the client before it is sent to the server (client only), returns the DB2 authorization ID associated with a given user.

GSS-API authentication
: Authentication that implements the required GSS-API functions, identifies the default security context (client side only), generates initial credentials based on user ID and password, and optionally changes password (client side only), creates and accepts security tickets, and returns the DB2 authorization ID associated with a given GSS-API security context.

The following are the definitions for terminology used in the descriptions of the plug-in APIs.

Plug-in
A dynamically loadable library that DB2 will load to access user-written authentication or group membership lookup functions.

Implicit authentication
A connection to a database without specifying a user ID or a password.

Explicit authentication
A connection to a database in which both the user ID and password are specified.

Authid
An internal ID representing an individual or group to which authorities and privileges within the database are granted. Internally, a DB2 authid is folded to upper-case and is a minimum of 8 characters (blank padded to 8 characters). Currently, DB2 requires authids, user IDs, passwords, group names, namespaces, and domain names that can be represented in 7-bit ASCII. The maximum length of an authid is 30 characters.

Local authorization
Authorization that is local to the server or client that implements it, that checks if a user is authorized to perform an action (other than connecting to the database), such as starting and stopping the database manager, turning DB2 trace on and off, or updating the database manager configuration.

Namespace
A collection or grouping of users within which individual user identifiers must be unique. Common examples include Windows domains and Kerberos Realms. For example, within the Windows domain ″usa.company.com″ all user names must be unique. For example, ″user1@usa.company.com″. The same user ID in another domain, as in the case of ″user1@canada.company.com″, however refers to a different person. A fully qualified user identifier includes a user ID and namespace pair; for example, ″user@domain.name″ or ″domain\user″.

Input
Indicates that DB2 will fill in the value for the security plug-in API parameter.

Output
Indicates that the security plug-in API will fill in the value for the API parameter.

Related concepts:

- “Security plug-ins” on page 1397

Related reference:

- “APIs for group retrieval plug-ins” on page 1427
- “APIs for user ID/password authentication plug-ins” on page 1436
- “Required APIs and definitions for GSS-API authentication plug-ins” on page 1460

Group plug-in APIs

APIs for group retrieval plug-ins

For the group retrieval plug-in library, you will need to implement the following APIs:

- `db2secGroupPluginInit`

  Note: The db2secGroupPluginInit API takes as input a pointer, `*logMessage_fn`, to an API with the following prototype:

  ```
  SQL_API_RC (SQL_API_FN db2secLogMessage)
  (
  db2int32 level,
  void    *data,
  db2int32 length
  );
  ```

  The db2secLogMessage API allows the plug-in to log messages to db2diag.log for debugging or informational purposes. This API is provided by DB2, so you need not implement it.
- `db2secPluginTerm`
- `db2secGetGroupsForUser`
- `db2secDoesGroupExist`
- `db2secFreeGroupListMemory`
- `db2secFreeErrormsg`
- The only API that must be resolvable externally is `db2secGroupPluginInit`. This API will take a `void *` parameter, which should be cast to the type:

  ```
  typedef struct db2secGroupFunctions_1
  {
  db2int32 version;
  db2int32 plugintype;
  SQL_API_RC (SQL_API_FN * db2secGetGroupsForUser)
  (
  const char *authid,
  db2int32    authidlen,
  const char *userid,
  db2int32    useridlen,
  const char *usernamespace,
  db2int32    usernamespacelen,
  db2int32    usernamespacetype,
  const char *dbname,
  db2int32    dbnamelen,
  const void *token,
  db2int32    tokentype,
  db2int32    location,
  const char *authpluginname,
  db2int32    authpluginnamelen,
  void      **grouplist,
  db2int32   *numgroups,
  char      **errormsg,
  db2int32   *errormsglen
  );

  SQL_API_RC (SQL_API_FN * db2secDoesGroupExist)
  (
  const char *groupname,
  db2int32    groupnamelen,
  char      **errormsg,
  db2int32   *errormsglen
  ```

```
);

SQL_API_RC (SQL_API_FN * db2secFreeGroupListMemory)
(
void     *ptr,
char    **errormsg,
db2int32 *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)
(
char *msgtobefree
);

SQL_API_RC (SQL_API_FN * db2secPluginTerm)
(
char    **errormsg,
db2int32 *errormsglen
);

} db2secGroupFunctions_1;
```

The `db2secGroupPluginInit` API assigns the addresses for the rest of the externally available functions.

Note: The _1 indicates that this is the structure corresponding to version 1 of the API. Subsequent interface versions will have the extension _2, _3, and so on.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related tasks:

- "Deploying a group retrieval plug-in" on page 1407

Related reference:

- "db2secDoesGroupExist - Check if group exists" on page 1434
- "db2secFreeErrormsg - Free error message memory" on page 1435
- "db2secFreeGroupListMemory - Free group list memory" on page 1435
- "db2secGetGroupsForUser - Get list of groups for user" on page 1430
- "db2secGroupPluginInit - Initialize group plug-in" on page 1428
- "db2secPluginTerm - Clean up group plug-in resources" on page 1430

db2secGroupPluginInit - Initialize group plug-in

Initialization API, for the group-retrieval plug-in, that DB2 calls immediately after loading the plug-in.

API and data structure syntax:

```
SQL_API_RC SQL_API_FN db2secGroupPluginInit
                              ( db2int32 version,
                                void *group_fns,
                                db2secLogMessage *logMessage_fn,
                                char     **errormsg,
                                db2int32 *errormsglen );
```

db2secGroupPluginInit API parameters:

version
Input. The highest version of the API supported by the instance loading that plugin. The value DB2SEC_API_VERSION (in db2secPlugin.h) contains the latest version number of the API that DB2 currently supports.

group_fns
Output. A pointer to the db2secGroupFunctions_<version_number> (also known as group_functions_<version_number>) structure. The db2secGroupFunctions_<version_number> structure contains pointers to the APIs implemented for the group-retrieval plug-in. In future, there might be different versions of the APIs (for example, db2secGroupFunctions_<version_number>), so the group_fns parameter is cast as a pointer to the db2secGroupFunctions_<version_number> structure corresponding to the version the plug-in has implemented. The first parameter of the group_functions_<version_number> structure tells DB2 the version of the APIs that the plug-in has implemented. Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented. The version number represents the version of the APIs implemented by the plugin, and the pluginType should be set to DB2SEC_PLUGIN_TYPE_GROUP.

logMessage_fn
Input. A pointer to the db2secLogMessage API, which is implemented by DB2. The db2secGroupPluginInit API can call the db2secLogMessage API to log messages to db2diag.log for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag.log file and the last two parameters respectively are the message string and its length. The valid values for the first parameter of dbesecLogMessage API (defined in db2secPlugin.h) are:

- `DB2SEC_LOG_NONE`: (0) No logging
- `DB2SEC_LOG_CRITICAL`: (1) Severe Error encountered
- `DB2SEC_LOG_ERROR`: (2) Error encountered
- `DB2SEC_LOG_WARNING`: (3) Warning
- `DB2SEC_LOG_INFO`: (4) Informational

The message text will show up in the diag.log only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter. So for example, if you use the `DB2SEC_LOG_INFO` value, the message text will only show up in the db2diag.log if the diaglevel database manager configuration parameter is set to 4.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGroupPluginInit API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for group retrieval plug-ins" on page 1427
- "Security plug-in samples" in *Samples Topics*

db2secPluginTerm - Clean up group plug-in resources

Frees resources used by the group-retrieval plug-in. This API is called by DB2 just before it unloads the group-retrieval plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform. For further information, please refer to the topic ″Restrictions on security plug-ins″.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secPluginTerm)
                      ( char      **errormsg,
                        db2int32 *errormsglen );
```

db2secPluginTerm API parameters:

errormsg
: Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secPluginTerm API execution is not successful.

errormsglen
: Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:
- "Restrictions on security plug-ins" on page 1413
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for group retrieval plug-ins" on page 1427
- "Security plug-in samples" in *Samples Topics*
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*

db2secGetGroupsForUser - Get list of groups for user

Returns the list of groups to which a user belongs.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGetGroupsForUser)
                      ( const char *authid,
                        db2int32 authidlen,
                        const char *userid,
                        db2int32 useridlen,
                        const char *usernamespace,
                        db2int32 usernamespacelen,
                        db2int32 usernamespacetype,
                        const char *dbname,
                        db2int32 dbnamelen,
                        void *token,
                        db2int32 tokentype,
                        db2int32 location,
                        const char *authpluginname,
                        db2int32 authpluginnamelen,
                        void      **grouplist,
                        db2int32 *numgroups,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secGetGroupsForUser API parameters:

authid Input. This parameter value is an SQL authid, which means that DB2 converts it to an uppercase character string with no trailing blanks. DB2 will always provide a non-null value for the authid parameter. The API must be able to return a list of groups to which the authid belongs without depending on the other input parameters. It is permissible to return a shortened or empty list if this cannot be determined.

If a user does not exist, the API must return the return code DB2SEC_PLUGIN_BADUSER. DB2 does not treat the case of a user not existing as an error, since it is permissible for an authid to not have any groups associated with it. For example, the db2secGetAuthids API can return an authid that does not exist on the operating system. The authid is not associated with any groups, however, it can still be assigned privileges directly.

If the API cannot return a complete list of groups using only the authid, then there will be some restrictions on certain SQL functions related to group support. For a list of possible problem scenarios, refer to the Usage notes section in this topic.

authidlen
Input. Length in bytes of the authid parameter value. DB2 will always provide a non-zero value for the authidlen parameter.

userid Input. This is the user ID corresponding to the authid. When this API is called on the server in a non-connect scenario, this parameter will not be filled by DB2.

useridlen
Input. Length in bytes of the userid parameter value.

usernamespace
Input. The namespace from which the user ID was obtained. When the user ID is not available, this parameter will not be filled by DB2.

usernamespacelen
Input. Length in bytes of the usernamespace parameter value.

usernamespacetype
Input. The type of namespace. Valid values for the usernamespacetype parameter (defined in db2secPlugin.h) are:

- DB2SEC_NAMESPACE_SAM_COMPATIBLE Corresponds to a username style like domain\myname
- DB2SEC_NAMESPACE_USER_PRINCIPAL Corresponds to a username style like myname@domain.ibm.com

Currently DB2 only supports the value DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, the usernamespacetype parameter is set to the value DB2SEC_USER_NAMESPACE_UNDEFINED (defined in db2secPlugin.h).

dbname
Input. Name of the database being connected to. This parameter can be NULL in a non-connect scenario.

dbnamelen
Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL in a non-connect scenario.

token
Input. A pointer to data provided by the authentication plug-in. It is not used by DB2. It provides the plug-in writer with the ability to coordinate user and group information. This parameter might not be provided in all cases (for example, in a non-connect scenario), in which case it will be NULL. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss_ctx_id_t).

tokentype
Input. Indicates the type of data provided by the authentication plug-in. If the authentication plug-in used is GSS-API based, the token will be set to the GSS-API context handle (gss_ctx_id_t). If the authentication plug-in used is user ID/password based, it will be a generic type. Valid values for the tokentype parameter (defined in db2secPlugin.h) are:

- `DB2SEC_GENERIC`: Indicates that the token is from a user ID/password based plug-in.
- `DB2SEC_GSSAPI_CTX_HANDLE`: Indicates that the token is from a GSS-API (including Kerberos) based plug-in.

location
Input. Indicates whether DB2 is calling this API on the client side or server side. Valid values for the location parameter (defined in db2secPlugin.h) are:

- `DB2SEC_SERVER_SIDE`: The API is to be called on the database server.
- `DB2SEC_CLIENT_SIDE`: The API is to be called on a client.

authpluginname
Input. Name of the authentication plug-in that provided the data in the token. The db2secGetGroupsForUser API might use this information in determining the correct group memberships. This parameter might not be filled by DB2 if the authid is not authenticated (for example, if the authid does not match the current connected user).

authpluginnamelen
Input. Length in bytes of the authpluginname parameter value.

grouplist
Output. List of groups to which the user belongs. The list of groups must be returned as a pointer to a section of memory allocated by the plug-in containing concatenated varchars (a varchar is a character array in which the first byte indicates the number of bytes following it). The length is an unsigned char (1 byte) and that limits the maximum length of a groupname to 255 characters. For example, ″\006GROUP1\007MYGROUP\008MYGROUP3″. Each group name should be a valid DB2 authid. The memory for this array must be allocated by the plug-in. The plug-in must therefore provide an API, such as the db2secFreeGroupListMemory API that DB2 will call to free the memory.

numgroups
Output. The number of groups contained in the grouplist parameter.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetGroupsForUser API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Usage notes:

The following is a list of scenarios when problems can occur if an incomplete group list is returned by this API to DB2:

- Embedded SQL application with DYNAMICRULES BIND (or DEFINEDBIND or INVOKEDBIND if the packages are running as a standalone application). DB2 checks for SYSADM membership and the application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- Alternate authorization is provided in CREATE SCHEMA statement. Group lookup will be performed against the AUTHORIZATION NAME parameter if there are nested CREATE statements in the CREATE SCHEMA statement.
- Embedded SQL applications with DYNAMICRULES DEFINERUN/DEFINEBIND and the packages are running in a routine context. DB2 checks for SYSADM membership of the routine definer and the application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- Processing a jar file in an MPP environment. In an MPP environment, the jar processing request is sent from the coordinator node with the session authid. The catalog node received the requests and process the jar files based on the privilege of the session authid (the user executing the jar processing requests).
  - Install jar file. The session authid needs to have one of the following rights: SYSADM, DBADM, or CREATEIN (implicit or explicit on the jar schema). The operation will fail if the above rights are granted to group containing the session authid, but not explicitly to the session authid or if only SYSADM is held, since SYSADM membership is determined by membership in the group defined by a database configuration parameter.
  - Remove jar file. The session authid needs to have one of the following rights rights: SYSADM, DBADM, or DROPIN (implicit or explicit on the jar schema), or is the definer of the jar file. The operation will fail if the above rights are granted to group containing the session authid, but not explicitly to the session authid, and if the session authid is not the definer of the jar file or if only SYSADM is held since SYSADM membership is determined by membership in the group defined by a database configuration parameter.
  - Replace jar file. This is same as removing the jar file, followed by installing the jar file. Both of the above apply.
- Regenerate views. This is triggered by the ALTER TABLE, ALTER COLUMN, SET DATA TYPE VARCHAR/VARGRAPHIC statements, or during migration. DB2 checks for SYSADM membership of the view definer. The application will fail if it is dependent on the implicit DBADM authority granted by being a member of the SYSADM group.
- When SET SESSION_USER statement is issued. Subsequent DB2 operations are run under the context of the authid specified by this statement. These operations

will fail if the privileges required are owned by one of the SESSION_USER's group is not explicitly granted to the SESSION_USER authid.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for group retrieval plug-ins" on page 1427
- "Security plug-in samples" in *Samples Topics*

db2secDoesGroupExist - Check if group exists

Determines if an authid represents a group. If the groupname exists, the API must be able to return the value DB2SEC_PLUGIN_OK, to indicate success. It must also be able to return the value DB2SEC_PLUGIN_INVALIDUSERORGROUP if the group name is not valid. It is permissible for the API to return the value DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN if it is impossible to determine if the input is a valid group. If an invalid group (DB2SEC_PLUGIN_INVALIDUSERORGROUP) or group not known (DB2SEC_PLUGIN_GROUPSTATUSNOTKNOWN) value is returned, DB2 might not be able to determine whether the authid is a group or user when issuing the GRANT statement without the keywords USER and GROUP, which would result in the error SQLCODE -569, SQLSTATE 56092 being returned to the user.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secDoesGroupExist)
                      ( const char *groupname,
                        db2int32 groupnamelen,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secDoesGroupExist API parameters:

groupname
: Input. An authid, upper-cased, with no trailing blanks.

groupnamelen
: Input. Length in bytes of the groupname parameter value.

errormsg
: Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesGroupExist API execution is not successful.

errormsglen
: Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for group retrieval plug-ins" on page 1427
- "Security plug-in samples" in *Samples Topics*

db2secFreeGroupListMemory - Free group list memory

Frees the memory used to hold the list of groups from a previous call to db2secGetGroupsForUser API.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeGroupListMemory)
                      ( void *ptr,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secFreeGroupListMemory API parameters:

ptr Input. Pointer to the memory to be freed.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeGroupListMemory API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for group retrieval plug-ins" on page 1427
- "Security plug-in samples" in *Samples Topics*

db2secFreeErrormsg - Free error message memory

Frees the memory used to hold an error message from a previous API call. This is the only API that does not return an error message. If this API returns an error, DB2 will log it and continue.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeErrormsg)
                      ( char *errormsg );
```

db2secFreeErrormsg API parameters:

msgtofree
Input. A pointer to the error message allocated from a previous API call.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for group retrieval plug-ins" on page 1427
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

User authentication plug-in APIs

APIs for user ID/password authentication plug-ins

For the user ID/password plug-in library, you will need to implement the following client-side APIs:

- `db2secClientAuthPluginInit`

  Note: The db2secClientAuthPluginInit API takes as input a pointer, `*logMessage_fn`, to an API with the following prototype:

  ```
  SQL_API_RC (SQL_API_FN db2secLogMessage)
  (
  db2int32 level,
  void    *data,
  db2int32 length
  );
  ```

  The db2secLogMessage API allows the plug-in to log messages to db2diag.log for debugging or informational purposes. This API is provided by DB2, so you need not implement it.
- `db2secClientAuthPluginTerm`
- `db2secGenerateInitialCred` (Only used for gssapi)
- `db2secRemapUserid` (Optional)
- `db2secGetDefaultLoginContext`
- `db2secValidatePassword`
- `db2secProcessServerPrincipalName` (This is only for GSS-API)
- `db2secFreeToken` (Functions to free memory held by the DLL)
- `db2secFreeErrormsg`
- `db2secFreeInitInfo`
- The only API that must be resolvable externally is `db2secClientAuthPluginInit`. This API will take a `void *` parameter, which should be cast to either:

  ```
  typedef struct db2secUseridPasswordClientAuthFunctions_1
  {
  db2int32 version;
  db2int32 plugintype;

  SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
  (
  char        authid[DB2SEC_MAX_AUTHID_LENGTH],
  db2int32   *authidlen,
  char        userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32   *useridlen,
  db2int32    useridtype,
  char        usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
  db2int32   *usernamespacelen,
  db2int32   *usernamespacetype,
  const char *dbname,
  db2int32    dbnamelen,
  void      **token,
  char      **errormsg,
  db2int32   *errormsglen
  );
  /* Optional */
  SQL_API_RC (SQL_API_FN * db2secRemapUserid)
  (
  char        userid[DB2SEC_MAX_USERID_LENGTH],
  db2int32   *useridlen,
  ```

```
char        usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
db2int32   *usernamespacelen,
db2int32   *usernamespacetype,
char        password[DB2SEC_MAX_PASSWORD_LENGTH],
db2int32   *passwordlen,
char        newpassword[DB2SEC_MAX_PASSWORD_LENGTH],
db2int32   *newpasswordlen,
const char *dbname,
db2int32    dbnamelen,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secValidatePassword)
(
const char *userid,
db2int32    useridlen,
const char *usernamespace,
db2int32    usernamespacelen,
db2int32    usernamespacetype,
const char *password,
db2int32    passwordlen,
const char *newpassword,
db2int32    newpasswordlen,
const char *dbname,
db2int32    dbnamelen,
db2Uint32   connection_details,
void      **token,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void     **token,
char     **errormsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)
(
char *errormsg
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char     **errormsg,
db2int32  *errormsglen
);
}
```

or

```
typedef struct db2secGssapiClientAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;

SQL_API_RC (SQL_API_FN * db2secGetDefaultLoginContext)
(
char        authid[DB2SEC_MAX_AUTHID_LENGTH],
db2int32   *authidlen,
char        userid[DB2SEC_MAX_USERID_LENGTH],
db2int32   *useridlen,
db2int32    useridtype,
char        usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
db2int32   *usernamespacelen,
db2int32   *usernamespacetype,
const char *dbname,
db2int32    dbnamelen,
void      **token,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secProcessServerPrincipalName)
(
const void *data,
gss_name_t *gssName,
char      **errormsg,
db2int32   *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secGenerateInitialCred)
(
const char    *userid,
db2int32       useridlen,
const char    *usernamespace,
db2int32       usernamespacelen,
db2int32       usernamespacetype,
const char    *password,
db2int32       passwordlen,
const char    *newpassword,
db2int32       newpasswordlen,
const char    *dbname,
db2int32       dbnamelen,
gss_cred_id_t *pGSSCredHandle,
void         **initInfo,
char         **errormsg,
db2int32      *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeToken)
(
void      *token,
char     **errormsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)
(
char *errormsg
);


SQL_API_RC (SQL_API_FN * db2secFreeInitInfo)
(
void      *initInfo,
char     **errormsg,
db2int32  *errormsglen
);

SQL_API_RC (SQL_API_FN * db2secClientAuthPluginTerm)
(
char     **errormsg,
db2int32  *errormsglen
);

/* GSS-API specific functions -- refer to db2secPlugin.h
   for parameter list*/

  OM_uint32 (SQL_API_FN * gss_init_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
  OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);
  }
```

You should use the `db2secUseridPasswordClientAuthFunctions_1` structure if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiClientAuthFunctions_1` structure.

For the user ID/password plug-in library, you will need to implement the following server-side APIs:

- `db2secServerAuthPluginInit`

  The db2secServerAuthPluginInit API takes as input a pointer, `*logMessage_fn`, to the db2secLogMessage API, and a pointer, `*getConDetails_fn`, to the db2secGetConDetails API with the following prototypes:

```
SQL_API_RC (SQL_API_FN db2secLogMessage)
(
db2int32 level,
void    *data,
db2int32 length
);

SQL_API_RC (SQL_API_FN db2secGetConDetails)
(
db2int32    conDetailsVersion,
const void *pConDetails
);
```

  The db2secLogMessage API allows the plug-in to log messages to db2diag.log for debugging or informational purposes. The db2secGetConDetails API allows the plug-in to obtain details about the client that is trying to attempt to have a database connection. Both the db2secLogMessage API and db2secGetConDetails API are provided by DB2. So you do not need to implement them. The db2secGetConDetails API in turn, takes as its second parameter,`pConDetails`, a pointer to either:

```
typedef struct db2sec_con_details_1
{
db2int32  clientProtocol;
db2Uint32 clientIPAddress;
db2Uint32 connect_info_bitmap;
db2int32  dbnameLen;
char      dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
} db2sec_con_details_1;
```

  or

```
typedef struct db2sec_con_details_2
{
  db2int32  clientProtocol;     /* See SQL_PROTOCOL_ in sqlenv.h */
  db2Uint32 clientIPAddress;    /* Set if protocol is TCPIP4     */
  db2Uint32 connect_info_bitmap;
  db2int32  dbnameLen;
  char dbname[DB2SEC_MAX_DBNAME_LENGTH + 1];
  db2Uint32 clientIP6Address[4];/* Set if protocol is TCPIP6     */
} db2sec_con_details_2;
```

  The possible values for `conDetailsVersion` are DB2SEC_CON_DETAILS_VERSION_1 and DB2SEC_CON_DETAILS_VERSION_2 representing the version of the API.

Note: While using db2sec_con_details_1 or db2sec_con_details_2, consider the following:

- Existing plugins that are using the db2sec_con_details_1 structure and the DB2SEC_CON_DETAILS_VERSION_1 value will continue to work as they did with v8.2 when calling the db2GetConDetails API. If this API is called on an IPv4 platform, the client IP address is returned in the clientIPAddress field of the structure. If this API is called on an IPv6 platform,a value of 0 is returned in the clientIPAddress field. To retrieve the client IP address on an IPv6 platform, the security plug-in code should be changed to use the db2sec_con_details_2 structure and the DB2SEC_CON_DETAILS_VERSION_2 value.
- New plugins should use the db2sec_con_details_2 structure and the DB2SEC_CON_DETAILS_VERSION_2 value. If the db2secGetConDetails API is called on an IPv4 platform, the client IP address is returned in the clientIPAddress field of the db2sec_con_details_2 structure and if the API is called on an IPv6 platform the client IP address is returned in the clientIP6Address field of the db2sec_con_details_2 structure. The *clientProtocol* field of the connection details structure will be set to one of SQL_PROTOCOL_TCPIP (IPv4, with v1 of the structure), SQL_PROTOCOL_TCPIP4 (IPv4, with v2 of the structure) or SQL_PROTOCOL_TCPIP6 (IPv6, with v2 of the structure).

- `db2secServerAuthPluginTerm`
- `db2secValidatePassword`
- `db2secGetAuthIDs`
- `db2secDoesAuthIDExist`
- `db2secFreeToken`
- `db2secFreeErrormsg`
- The only API that must be resolvable externally is `db2secServerAuthPluginInit`. This API will take a `void *` parameter, which should be cast to either:

```
typedef struct db2secUseridPasswordServerAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;

  /* parameter lists left blank for readability
    see above for parameters */
SQL_API_RC (SQL_API_FN * db2secValidatePassword)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeToken)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
} userid_password_server_auth_functions;
```

or

```
typedef struct db2secGssapiServerAuthFunctions_1
{
db2int32 version;
db2int32 plugintype;
gss_buffer_desc serverPrincipalName;
gss_cred_id_t ServerCredHandle;
SQL_API_RC (SQL_API_FN * db2secGetAuthIDs)(<parameter list);
SQL_API_RC (SQL_API_FN * db2secDoesAuthIDExist)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secFreeErrormsg)(<parameter list>);
SQL_API_RC (SQL_API_FN * db2secServerAuthPluginTerm)();
/* GSS-API specific functions
refer to db2secPlugin.h for parameter list*/
OM_uint32 (SQL_API_FN * gss_accept_sec_context )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_display_name )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_delete_sec_context )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_display_status )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_buffer )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_cred )(<parameter list>);
OM_uint32 (SQL_API_FN * gss_release_name )(<parameter list>);

} gssapi_server_auth_functions;
```

You should use the `db2secUseridPasswordServerAuthFunctions_1` structure if you are writing an user ID/password plug-in. If you are writing a GSS-API (including Kerberos) plug-in, you should use the `db2secGssapiServerAuthFunctions_1` structure.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related tasks:

- "Deploying a user ID/password plug-in" on page 1408

Related reference:

- "db2secClientAuthPluginInit - Initialize client authentication plug-in" on page 1441
- "db2secClientAuthPluginTerm - Clean up client authentication plug-in resources" on page 1443
- "db2secGetGroupsForUser - Get list of groups for user" on page 1430
- "db2secDoesAuthIDExist - Check if authentication ID exists" on page 1459
- "db2secFreeErrormsg - Free error message memory" on page 1435
- "db2secFreeInitInfo - Clean up resources held by the db2secGenerateInitialCred" on page 1453
- "db2secFreeToken - Free memory held by token" on page 1452
- "db2secGenerateInitialCred - Generate initial credentials" on page 1447
- "db2secGetAuthIDs - Get authentication IDs" on page 1457
- "db2secGetDefaultLoginContext - Get default login context" on page 1446
- "db2secProcessServerPrincipalName - Process service principal name returned from server" on page 1452
- "db2secRemapUserid - Remap user ID and password" on page 1444
- "db2secServerAuthPluginInit - Initialize server authentication plug-in" on page 1454
- "db2secServerAuthPluginTerm - Clean up server authentication plug-in resources" on page 1456
- "db2secValidatePassword - Validate password" on page 1449

db2secClientAuthPluginInit - Initialize client authentication plug-in

Initialization API, for the client authentication plug-in, that DB2 calls immediately after loading the plug-in.

db2secClientAuthPluginInit - Initialize client authentication plug-in

API and data structure syntax:

```
SQL_API_RC SQL_API_FN db2secClientAuthPluginInit
                              ( db2int32 version,
                                void *client_fns,
                                db2secLogMessage *logMessage_fn,
                                char    **errormsg,
                                db2int32 *errormsglen );
```

db2secClientAuthPluginInit API parameters:

version
: Input. The highest version number of the API that DB2 currently supports. The DB2SEC_API_VERSION value (in db2secPlugin.h) contains the latest version number of the API that DB2 currently supports.

client_fns
: Output. A pointer to memory provided by DB2 for a db2secGssapiClientAuthFunctions_<version_number> structure (also known as gssapi_client_auth_functions_<version_number>), if GSS-API authentication is used, or a db2secUseridPasswordClientAuthFunctions_<version_number> structure (also known as userid_password_client_auth_functions_<version_number>), if userid/password authentication is used. The db2secGssapiClientAuthFunctions_<version_number> structure and db2secUseridPasswordClientAuthFunctions_<version_number> structure respectively contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/password authentication plug-in. In future versions of DB2, there might be different versions of the APIs, so the client_fns parameter is cast as a pointer to the gssapi_client_auth_functions_<version_number> structure corresponding to the version the plug-in has implemented.

The first parameter of the gssapi_client_auth_functions_<version_number> structure or the userid_password_client_auth_functions_<version_number> structure tells DB2 the version of the APIs that the plug-in has implemented.

Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented.

Inside the gssapi_server_auth_functions_<version_number> or userid_password_server_auth_functions_<version_number> structure, the plugintype parameter should be set to one of DB2SEC_PLUGIN_TYPE_USERID_PASSWORD, DB2SEC_PLUGIN_TYPE_GSSAPI, or DB2SEC_PLUGIN_TYPE_KERBEROS. Other values can be defined in future versions of the API.

logMessage_fn
: Input. A pointer to the db2secLogMessage API, which is implemented by DB2. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to db2diag.log for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag.log file and the last two parameters respectively are the message string and its length. The valid values for the first parameter of dbesecLogMessage API (defined in db2secPlugin.h) are:

- DB2SEC_LOG_NONE (0) No logging

- DB2SEC_LOG_CRITICAL (1) Severe Error encountered
- DB2SEC_LOG_ERROR (2) Error encountered
- DB2SEC_LOG_WARNING (3) Warning
- DB2SEC_LOG_INFO (4) Informational

The message text will show up in the diag.log only if the value of the ′level′ parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter. So for example, if you use the DB2SEC_LOG_INFO value, the message text will only show up in the db2diag.log if the diaglevel database manager configuration parameter is set to 4.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secClientAuthPluginInit API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:
- “Security plug-in APIs” on page 1425
- “Security plug-ins” on page 1397

Related reference:
- “APIs for user ID/password authentication plug-ins” on page 1436
- “Security plug-in samples” in *Samples Topics*
- “diaglevel - Diagnostic error capture level configuration parameter” in *Performance Guide*

db2secClientAuthPluginTerm - Clean up client authentication plug-in resources

Frees resources used by the client authentication plug-in. This API is called by DB2 just before it unloads the client authentication plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform. For further information, please refer to the topic ″Restrictions on security plug-ins″.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secClientAuthPluginTerm)
                      ( char     **errormsg,
                        db2int32 *errormsglen);
```

db2secClientAuthPluginTerm API parameters:

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secClientAuthPluginTerm API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Restrictions on security plug-ins" on page 1413
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

db2secRemapUserid - Remap user ID and password

This API is called by DB2 on the client side to provide the ability to remap a given user ID and password (and possibly new password and usernamespace) to values different from those given at connect time. DB2 will only call this API if a user ID and a password are supplied at connect time. This prevents a plug-in from remapping a user ID by itself to a user ID/password pair. This API is optional and will not be called if it is not provided or implemented by the security plug-in.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secRemapUserid)
                        ( char userid[DB2SEC_MAX_USERID_LENGTH],
                          db2int32 *useridlen,
                          char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
                          db2int32 *usernamespacelen,
                          db2int32 *usernamespacetype,
                          char password[DB2SEC_MAX_PASSWORD_LENGTH],
                          db2int32 *passwordlen,
                          char newpasswd[DB2SEC_MAX_PASSWORD_LENGTH],
                          db2int32 *newpasswdlen,
                          const char *dbname,
                          db2int32 dbnamelen,
                          char     **errormsg,
                          db2int32 *errormsglen);
```

db2secRemapUserid API parameters:

userid Input or output. The user ID to be remapped. If there is an input user ID value, then the API must provide an output user ID value that can be the same or different from the input user ID value. If there is no input user ID value, then the API should not return an output user ID value.

useridlen
Input or output. Length in bytes of the userid parameter value.

usernamespace
Input or output. The namespace of the user ID. This value can optionally be remapped. If no input parameter value is specified, but an output value is returned, then the usernamespace will only be used by DB2 for CLIENT type authentication and will be disregarded for other authentication types.

usernamespacelen
Input or output. Length in bytes of the usernamespace parameter value. Under the limitation that the usernamespacetype parameter must be set to

the value DB2SEC_NAMESPACE_SAM_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

usernamespacetype
Input or output. Old and new namespacetype value. In the current version of DB2, the only supported namespace type value is DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponds to a username style like domain\myname).

password
Input or output. As an input, it is the password that is to be remapped. As an output it is the remapped password. If an input value is specified for this parameter, the API must be able to return an output value that differs from the input value. If no input value is specified, the API must not return an output password value.

passwordlen
Input or output. Length in bytes of the password parameter value.

newpasswd
Input or output. As an input, it is the new password that is to be set. As an output it is the confirmed new password.

Note: This is the new password that will be passed by DB2 into the newpassword parameter of the db2secValidatePassword API on the client or the server (depending on the value of the authentication database manager configuration parameter). If a new password was passed as input, then the API must be able to return an output value and it can be a different new password. If there is no new password passed in as input, then the API should not return an output new password.

newpasswdlen
Input or output. Length in bytes of the newpasswd parameter value.

dbname
Input. Name of the database to which the client is connecting.

dbnamelen
Input. Length in bytes of the dbname parameter value.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secRemapUserid API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- “Security plug-in APIs” on page 1425
- “Security plug-ins” on page 1397

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 1436
- “Security plug-in samples” in *Samples Topics*

db2secGetDefaultLoginContext - Get default login context

Determines the user associated with the default login context, in other words, determines the DB2 authid of the user invoking a DB2 command without explicitly specifying a user ID (either an implicit authentication to a database, or a local authorization). This API must return both an authid and a user ID.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGetDefaultLoginContext)
                      ( char authid[DB2SEC_MAX_AUTHID_LENGTH],
                        db2int32 *authidlen,
                        char userid[DB2SEC_MAX_USERID_LENGTH],
                        db2int32 *useridlen,
                        db2int32 useridtype,
                        char usernamespace[DB2SEC_MAX_USERNAMESPACE_LENGTH],
                        db2int32 *usernamespacelen,
                        db2int32 *usernamespacetype,
                        const char *dbname,
                        db2int32 dbnamelen,
                        void      **token,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secGetDefaultLoginContext API parameters:

authid
Output. The parameter in which the authid should be returned. The returned value must conform to DB2 authid naming rules, or the user will not be authorized to perform the requested action.

authidlen
Output. Length in bytes of the authid parameter value.

userid
Output. The parameter in which the user ID associated with the default login context should be returned.

useridlen
Output. Length in bytes of the userid parameter value.

useridtype
Input. Indicates if the real or effective user ID of the process is being specified. On Windows, only the real user ID exists. On UNIX and Linux, the real user ID and effective user ID can be different if the `uid` user ID for the application is different than the ID of the user executing the process. Valid values for the userid parameter (defined in db2secPlugin.h) are:

DB2SEC_PLUGIN_REAL_USER_NAME
Indicates that the real user ID is being specified.

DB2SEC_PLUGIN_EFFECTIVE_USER_NAME
Indicates that the effective user ID is being specified.

Note: Some plug-in implementations might not distinguish between the real and effective userid. In particular, a plug-in that does not use the UNIX or Linux identity of the user to establish the DB2 AUTHID can safely ignore this distinction.

usernamespace
Output. The namespace of the user ID.

usernamespacelen
Output. Length in bytes of the usernamespace parameter value. Under the limitation that the usernamespacetype parameter must be set to the value

DB2SEC_NAMESPACE_SAM_COMPATIBLE (defined in db2secPlugin.h), the maximum length currently supported is 15 bytes.

usernamespacetype
Output. Namespacetype value. In the current version of DB2, the only supported namespace type is DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponds to a username style like domain\myname).

dbname
Input. Contains the name of the database being connected to, if this call is being used in the context of a database connection. For local authorization actions or instance attachments, this parameter is set to NULL.

dbnamelen
Input. Length in bytes of the dbname parameter value.

token
Output. This is a pointer to data allocated by the plug-in that it might pass to subsequent authentication calls in the plug-in, or possibly to the group retrieval plug-in. The structure of this data is determined by the plug-in writer.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetDefaultLoginContext API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*
- "authentication - Authentication type " on page 1497

db2secGenerateInitialCred - Generate initial credentials

Obtains the initial GSS-API credentials based on the user ID and password that are passed in. For Kerberos, this will be the ticket-granting ticket (TGT). The credential handle that is returned in pGSSCredHandle parameter is the handle that is used with the gss_init_sec_context API and must be either an INITIATE or BOTH credential. The db2secGenerateInitialCred API is only called when a user ID, and possibly a password are supplied. Otherwise, DB2 will specify the value GSS_C_NO_CREDENTIAL when calling the gss_init_sec_context API to signify that the default credential obtained from the current login context is to be used.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGenerateInitialCred)
                      ( const char *userid,
                        db2int32 useridlen,
                        const char *usernamespace,
                        db2int32 usernamespacelen,
                        db2int32 usernamespacetype,
                        const char *password,
                        db2int32 passwordlen,
                        const char *newpassword,
                        db2int32 newpasswordlen,
                        const char *dbname,
                        db2int32 dbnamelen,
                        gss_cred_id_t *pGSSCredHandle,
                        void         **InitInfo,
                        char         **errormsg,
                        db2int32 *errormsglen );
```

db2secGenerateInitialCred API parameters:

userid Input. The user ID whose password is to be verified on the database server.

useridlen
Input. Length in bytes of the userid parameter value.

usernamespace
Input. The namespace from which the user ID was obtained.

usernamespacelen
Input. Length in bytes of the usernamespace parameter value.

usernamespacetype
Input. The type of namespace.

password
Input. The password to be verified.

passwordlen
Input. Length in bytes of the password parameter value.

newpassword
Input. A new password if the password is to be changed. If no change is requested, the newpassword parameter is set to NULL. If it is not NULL, the API should validate the old password before setting the password to its new value. The API does not have to honour a request to change the password, but if it does not, it should immediately return with the return value DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED without validating the old password.

newpasswordlen
Input. Length in bytes of the newpassword parameter value.

dbname
Input. The name of the database being connected to. The API is free to ignore this parameter, or the API can return the value DB2SEC_PLUGIN_CONNECTION_DISALLOWED if it has a policy of restricting access to certain databases to users who otherwise have valid passwords.

dbnamelen
Input. Length in bytes of the dbname parameter value.

pGSSCredHandle
Output. Pointer to the GSS-API credential handle.

InitInfo
Output. A pointer to data that is not known to DB2. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. DB2 will call the db2secFreeInitInfo API at the end of the authentication process, at which point these resources

are freed. If the db2secGenerateInitialCred API does not need to maintain such a list, then it should return NULL.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGenerateInitialCred API execution is not successful.

Note: For this API, error messages should not be created if the return value indicates a bad user ID or password. An error message should only be returned if there is an internal error in the API that prevented it from completing properly.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397
- "Authentication methods for your server" on page 89

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

db2secValidatePassword - Validate password

Provides a method for performing user ID and password style authentication during a database connect operation.

Note: When the API is run on the client side, the API code is run with the privileges of the user executing the CONNECT statement. This API will only be called on the client side if the authentication configuration parameter is set to CLIENT.

When the API is run on the server side, the API code is run with the privileges of the instance owner.

The plug-in writer should take the above into consideration if authentication requires special privileges (such as root level system access on UNIX).

This API must return the value DB2SEC_PLUGIN_OK (success) if the password is valid, or an error code such as DB2SEC_PLUGIN_BADPWD if the password is invalid.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secValidatePassword)
                      ( const char *userid,
                        db2int32 useridlen,
                        const char *usernamespace,
                        db2int32 usernamespacelen,
                        db2int32 usernamespacetype,
                        const char *password,
                        db2int32 passwordlen,
                        const char *newpasswd,
                        db2int32 newpasswdlen,
                        const char *dbname,
                        db2int32 dbnamelen,
                        db2Uint32 connection_details,
                        void      **token,
                        char      **errormsg,
                        db2int32 *errormsglen );
```

db2secValidatePassword API parameters:

userid Input. The user ID whose password is to be verified.

useridlen
Input. Length in bytes of the userid parameter value.

usernamespace
Input. The namespace from which the user ID was obtained.

usernamespacelen
Input. Length in bytes of the usernamespace parameter value.

usernamespacetype
Input. The type of namespace. Valid values for the usernamespacetype parameter (defined in db2secPlugin.h) are:

- `DB2SEC_NAMESPACE_SAM_COMPATIBLE` Corresponds to a username style like domain\myname
- `DB2SEC_NAMESPACE_USER_PRINCIPAL` Corresponds to a username style like myname@domain.ibm.com

Currently DB2 only supports the value DB2SEC_NAMESPACE_SAM_COMPATIBLE. When the user ID is not available, the usernamespacetype parameter is set to the value DB2SEC_USER_NAMESPACE_UNDEFINED (defined in db2secPlugin.h).

password
Input. The password to be verified.

passwordlen
Input. Length in bytes of the password parameter value.

newpasswd
Input. A new password, if the password is to be changed. If no change is requested, this parameter is set to NULL. If this parameter is not NULL, the API should validate the old password before changing it to the new password. The API does not have to fulfill a request to change the password, but if it does not, it should immediately return with the return value DB2SEC_PLUGIN_CHANGEPASSWORD_NOTSUPPORTED without validating the old password.

newpasswdlen
Input. Length in bytes of the newpasswd parameter value.

dbname
Input. The name of the database being connected to. The API is free to ignore the dbname parameter, or it can return the value DB2SEC_PLUGIN_CONNECTIONREFUSED if it has a policy of restricting access to certain databases to users who otherwise have valid passwords. This parameter can be NULL.

dbnamelen
Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL.

connection_details
Input. A 32-bit parameter of which 3 bits are currently used to store the following information:

- The right-most bit indicates whether the source of the user ID is the default from the db2secGetDefaultLoginContext API, or was explicitly provided during the connect.
- The second-from-right bit indicates whether the connection is local (using Inter Process Communication (IPC) or a connect from one of the nodes in the db2nodes.cfg in the partitioned database environment), or remote (through a network or loopback). This gives the API the ability to decide whether clients on the same machine can connect to the DB2 server without a password. Due to the default OS-based user ID/password plugin, local connections are permitted without a password from clients on the same machine (assuming the user has connect privileges).
- The third-from-right bit indicates whether DB2 is calling the API from the server side or client side.

The bit values are defined in db2secPlugin.h:

- `DB2SEC_USERID_FROM_OS` (0x00000001) Indicates that the user ID is obtained from OS and not explicitly given on the connect statement.
- `DB2SEC_CONNECTION_ISLOCAL` (0x00000002) Indicates a local connection.
- `DB2SEC_VALIDATING_ON_SERVER_SIDE` (0x0000004) Indicates whether DB2 is calling from the server side or client side to validate password. If this bit value is set, then DB2 is calling from server side, otherwise it is calling from the client side.

DB2's default behavior for an implicit authentication is to allow the connection without any password validation. However, plug-in developers have the option to disallow implicit authentication by returning a DB2SEC_PLUGIN_BADPASSWORD error.

token Input. A pointer to data which can be passed as a parameter to subsequent API calls during the current connection. Possible APIs that might be called include db2secGetAuthIDs API and db2secGetGroupsForUser API.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secValidatePassword API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "Security plug-in samples" in *Samples Topics*
- "APIs for user ID/password authentication plug-ins" on page 1436

db2secProcessServerPrincipalName - Process service principal name returned from server

Processes the service principal name returned from the server and returns the principal name in the gss_name_t internal format to be used with the gss_init_sec_context API. The db2secProcessServerPrincipalName API also processes the service principal name cataloged with the database directory when Kerberos authentication is used. Ordinarily, this conversion uses the gss_import_name API. After the context is established, the gss_name_t object is freed through the call to gss_release_name API. The db2secProcessServerPrincipalName API returns the value DB2SEC_PLUGIN_OK if gssName parameter points to a valid GSS name; a DB2SEC_PLUGIN_BAD_PRINCIPAL_NAME error code is returned if the principal name is invalid.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secProcessServerPrincipalName)
                      ( const char *name,
                        db2int32 namelen,
                        gss_name_t *gssName,
                        char       **errormsg,
                        db2int32 *errormsglen );
```

db2secProcessServerPrincipalName API parameters:

name
Input. Text name of the service principal in GSS_C_NT_USER_NAME format; for example, service/host@REALM.

namelen
Input. Length in bytes of the name parameter value.

gssName
Output. Pointer to the output service principal name in the GSS-API internal format.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secProcessServerPrincipalName API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

db2secFreeToken - Free memory held by token

Frees the memory held by token. This API is called by DB2 when it no longer needs the memory held by the token parameter.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeToken)
                       ( void *token,
                         char      **errormsg,
                         db2int32 *errormsglen );
```

db2secFreeToken API parameters:

token Input. Pointer to the memory to be freed.

errormsg
: Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeToken API execution is not successful.

errormsglen
: Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

db2secFreeInitInfo - Clean up resources held by the db2secGenerateInitialCred

Frees any resources allocated by the db2secGenerateInitialCred API. This can include, for example, handles to underlying mechanism contexts or a credential cache created for the GSS-API credential cache.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secFreeInitInfo)
                       ( void *initinfo,
                         char     **errormsg,
                         db2int32 *errormsglen);
```

db2secFreeInitInfo API parameters:

initinfo
: Input. A pointer to data that is not known to DB2. The plug-in can use this memory to maintain a list of resources that are allocated in the process of generating the credential handle. These resources are freed by calling this API.

errormsg
: Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secFreeInitInfo API execution is not successful.

errormsglen
: Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

db2secServerAuthPluginInit - Initialize server authentication plug-in

Initialization API, for the server authentication plug-in, that DB2 calls immediately after loading the plug-in. In the case of GSS-API, the plug-in is responsible for filling in the server's principal name in the serverPrincipalName parameter inside the gssapi_server_auth_functions structure at initialization time and providing the server's credential handle in the serverCredHandle parameter inside the gssapi_server_auth_functions structure. The freeing of the memory allocated to hold the principal name and the credential handle must be done by the db2secServerAuthPluginTerm API by calling the gss_release_name and gss_release_cred APIs.

API and data structure syntax:

```
SQL_API_RC SQL_API_FN db2secServerAuthPluginInit
                              ( db2int32 version,
                                void *server_fns,
                                db2secGetConDetails *getConDetails_fn,
                                db2secLogMessage *logMessage_fn,
                                char              **errormsg,
                                db2int32 *errormsglen );
```

db2secServerAuthPluginInit API parameters:

version
: Input. The highest version number of the API that DB2 currently supports. The DB2SEC_API_VERSION value (in db2secPlugin.h) contains the latest version number of the API that DB2 currently supports.

server_fns
: Output. A pointer to memory provided by DB2 for a db2secGssapiServerAuthFunctions_<version_number> structure (also known as gssapi_server_auth_functions_<version_number>), if GSS-API authentication is used, or a db2secUseridPasswordServerAuthFunctions_<version_number> structure (also known as userid_password_server_auth_functions_<version_number>), if userid/password authentication is used. The db2secGssapiServerAuthFunctions_<version_number> structure and db2secUseridPasswordServerAuthFunctions_<version_number> structure respectively contain pointers to the APIs implemented for the GSS-API authentication plug-in and userid/password authentication plug-in.

  In future versions of DB2, there might be different versions of the APIs, so the server_fns parameter is cast as a pointer to the gssapi_server_auth_functions_<version_number> structure corresponding to the version the plug-in has implemented. The first parameter of the gssapi_server_auth_functions_<version_number> structure or the userid_password_server_auth_functions_<version_number> structure tells DB2 the version of the APIs that the plug-in has implemented.

Note: The casting is done only if the DB2 version is higher or equal to the version of the APIs that the plug-in has implemented.

Inside the gssapi_server_auth_functions_<version_number> or userid_password_server_auth_functions_<version_number> structure, the plugintype parameter should be set to one of DB2SEC_PLUGIN_TYPE_USERID_PASSWORD, DB2SEC_PLUGIN_TYPE_GSSAPI, or DB2SEC_PLUGIN_TYPE_KERBEROS. Other values can be defined in future versions of the API.

getConDetails_fn

Input. Pointer to the db2secGetConDetails API, which is implemented by DB2. The db2secServerAuthPluginInit API can call the db2secGetConDetails API in any one of the other authentication APIs to obtain details related to the database connection. These details include information about the communication mechanism associated with the connection (such as the IP address, in the case of TCP/IP), which the plug-in writer might need to reference when making authentication decisions. For example, the plug-in could disallow a connection for a particular user, unless that user is connecting from a particular IP address. The use of the db2secGetConDetails API is optional.

If the db2secGetConDetails API is called in a situation not involving a database connection, it returns the value DB2SEC_PLUGIN_NO_CON_DETAILS, otherwise, it returns 0 on success.

The db2secGetConDetails API takes two input parameters; pConDetails, which is a pointer to the db2sec_con_details_<version_number> structure, and conDetailsVersion, which is a version number indicating which db2sec_con_details structure to use. Possible values are DB2SEC_CON_DETAILS_VERSION_1 when db2sec_con_details1 is used or DB2SEC_CON_DETAILS_VERSION_2 when db2sec_con_details2. The recommended version number to use is DB2SEC_CON_DETAILS_VERSION_2.

Upon a successful return, the db2sec_con_details structure (either db2sec_con_details1 or db2sec_con_details2) will contain the following information:

- The protocol used for the connection to the server. The listing of protocol definitions can be found in the file sqlenv.h (located in the include directory) (SQL_PROTOCOL_*). This information is filled out in the clientProtocol parameter.
- The TCP/IP address of the inbound connect to the server if the clientProtocol is SQL_PROTOCOL_TCPIP or SQL_PROTOCOL_TCPIP4. This information is filled out in the clientIPAddress parameter.
- The database name the client is attempting to connect to. This will not be set for instance attachments. This information is filled out in the dbname and dbnameLen parameters.
- A connection information bit-map that contains the same details as documented in the connection_details parameter of the db2secValidatePassword API. This information is filled out in the connect_info_bitmap parameter.
- The TCP/IP address of the inbound connect to the server if the clientProtocol is SQL_PROTOCOL_TCPIP6. This information is filled out in the clientIP6Address parameter and it is only available if DB2SEC_CON_DETAILS_VERSION_2 is used for db2secGetConDetails API call.

logMessage_fn

Input. A pointer to the db2secLogMessage API, which is implemented by DB2. The db2secClientAuthPluginInit API can call the db2secLogMessage API to log messages to db2diag.log for debugging or informational purposes. The first parameter (level) of db2secLogMessage API specifies the type of diagnostic errors that will be recorded in the db2diag.log file and the last two parameters respectively are the message string and its length. The valid values for the first parameter of dbesecLogMessage API (defined in db2secPlugin.h) are:

DB2SEC_LOG_NONE (0)
No logging

DB2SEC_LOG_CRITICAL (1)
Severe Error encountered

DB2SEC_LOG_ERROR (2)
Error encountered

DB2SEC_LOG_WARNING (3)
Warning

DB2SEC_LOG_INFO (4)
Informational

The message text will show up in the diag.log only if the value of the 'level' parameter of the db2secLogMessage API is less than or equal to the diaglevel database manager configuration parameter.

So for example, if you use the DB2SEC_LOG_INFO value, the message text will only show up in the db2diag.log if the diaglevel database manager configuration parameter is set to 4.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secServerAuthPluginInit API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*
- "diaglevel - Diagnostic error capture level configuration parameter" in *Performance Guide*

db2secServerAuthPluginTerm - Clean up server authentication plug-in resources

Frees resources used by the server authentication plug-in. This API is called by DB2 just before it unloads the server authentication plug-in. It should be implemented in a manner that it does a proper cleanup of any resources the plug-in library holds, for instance, free any memory allocated by the plug-in, close files that are still open, and close network connections. The plug-in is responsible for keeping track of these resources in order to free them. This API is not called on any Windows platform. For further information, please refer to the topic ″Restrictions on security plug-ins″.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secServerAuthPluginTerm)
                       ( char     **errormsg,
                         db2int32 *errormsglen );
```

db2secServerAuthPluginTerm API parameters:

errormsg
: Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secServerAuthPluginTerm API execution is not successful.

errormsglen
: Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- “Security plug-in APIs” on page 1425
- “Restrictions on security plug-ins” on page 1413
- “Security plug-ins” on page 1397

Related reference:

- “APIs for user ID/password authentication plug-ins” on page 1436
- “Security plug-in samples” in *Samples Topics*

db2secGetAuthIDs - Get authentication IDs

Returns an SQL authid for an authenticated user. This API is called during database connections for both user ID/password and GSS-API authentication methods.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secGetAuthIDs)
                       ( const char *userid,
                         db2int32 useridlen,
                         const char *usernamespace,
                         db2int32 usernamespacelen,
                         db2int32 usernamespacetype,
                         const char *dbname,
                         db2int32 dbnamelen,
                         void      **token,
                         char SystemAuthID[DB2SEC_MAX_AUTHID_LENGTH],
                         db2int32 *SystemAuthIDlen,
                         char InitialSessionAuthID[DB2SEC_MAX_AUTHID_LENGTH],
                         db2int32 *InitialSessionAuthIDlen,
                         char username[DB2SEC_MAX_USERID_LENGTH],
                         db2int32 *usernamelen,
                         db2int32 *initsessionidtype,
                         char      **errormsg,
                         db2int32 *errormsglen );
```

db2secGetAuthIDs API parameters:

userid
Input. The authenticated user. This is blank for GSS-API authentication method.

useridlen
Input. Length in bytes of the userid parameter value.

usernamespace
Input. The namespace from which the user ID was obtained.

usernamespacelen
Input. Length in bytes of the usernamespace parameter value.

usernamespacetype
Input. Namespacetype value. In the current version of DB2, the only supported namespace type value is DB2SEC_NAMESPACE_SAM_COMPATIBLE (corresponds to a username style like domain\myname).

dbname
Input. The name of the database being connected to. The API can ignore this, or it can return differing authids when the same user connects to different databases. This parameter can be NULL.

dbnamelen
Input. Length in bytes of the dbname parameter value. This parameter is set to 0 if dbname parameter is NULL.

token
Input or output. Data that the plug-in might pass to the db2secGetGroupsForUser API. For GSS-API, this is a context handle (gss_ctx_id_t). Ordinarily, token is an input-only parameter and its value is taken from the db2secValidatePassword API. It can also be an output parameter when authentication is done on the client and therefore db2secValidatePassword API is not called.

SystemAuthID
Output. The system authid that corresponds to the ID of the authenticated user. The size is 255 bytes, but DB2 currently uses only up to (and including) 30 bytes.

SystemAuthIDlen
Output. Length in bytes of the SystemAuthID parameter value.

InitialSessionAuthID
Output. Authid used for this connection session. This is usually the same as the SystemAuthID parameter but can be different in some situations, for instance, when issuing a SET SESSION AUTHORIZATION statement. The size is 255 bytes, but DB2 currently uses only up to (and including) 30 bytes.

InitialSessionAuthIDlen
Output. Length in bytes of the InitialSessionAuthID parameter value.

username
Output. A username corresponding to the authenticated user and authid. This will only be used for auditing and will be logged in the "User ID" field in the audit record for CONNECT statement. If the API does not fill in the username parameter, DB2 copies it from the userid.

usernamelen
Output. Length in bytes of the username parameter value.

initsessionidtype
Output. Session authid type indicating whether or not the InitialSessionAuthid parameter is a role or an authid. The API should return one of the following values (defined in db2secPlugin.h):

- DB2SEC_ID_TYPE_AUTHID (0)
- DB2SEC_ID_TYPE_ROLE (1)

Currently, DB2 only supports authid (DB2SEC_ID_TYPE_AUTHID).

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secGetAuthIDs API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length in bytes of the error message string in errormsg parameter.

Related concepts:

- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:

- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

db2secDoesAuthIDExist - Check if authentication ID exists

Determines if the authid represents an individual user (for instance, whether the API can map the authid to an external user id). The API should return the value DB2SEC_PLUGIN_OK if it is successful - the authid is valid, DB2SEC_PLUGIN_INVALID_USERORGROUP if it is not valid, or DB2SEC_PLUGIN_USERSTATUSNOTKNOWN if the authid existence cannot be determined.

API and data structure syntax:

```
SQL_API_RC ( SQL_API_FN *db2secDoesAuthIDExist)
                       ( const char *authid,
                         db2int32 authidlen,
                         char      **errormsg,
                         db2int32 *errormsglen );
```

db2secDoesAuthIDExist API parameters:

authid Input. The authid to validate. This is upper-cased, with no trailing blanks.

authidlen
Input. Length in bytes of the authid parameter value.

errormsg
Output. A pointer to the address of an ASCII error message string allocated by the plug-in that can be returned in this parameter if the db2secDoesAuthIDExist API execution is not successful.

errormsglen
Output. A pointer to an integer that indicates the length of the error message string in errormsg parameter.

Related concepts:
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Related reference:
- "APIs for user ID/password authentication plug-ins" on page 1436
- "Security plug-in samples" in *Samples Topics*

GSS-API plug-in APIs

Required APIs and definitions for GSS-API authentication plug-ins

Following is a complete list of GSS-APIs required for the DB2 security plug-in interface. The supported APIs follow these specifications: *Generic Security Service Application Program Interface, Version 2* (IETF RFC2743) and *Generic Security Service API Version 2: C-Bindings* (IETF RFC2744). Before implementing a GSS-API based plug-in, you should have a complete understanding of these specifications.

Table 162. Required APIs and Definitions for GSS-API authentication plug-ins

Name		Description
Client-side APIs	`gss_init_sec_context`	Initiate a security context with a peer application.
Server-side APIs	`gss_accept_sec_context`	Accept a security context initiated by a peer application.
Server-side APIs	`gss_display_name`	Convert an internal format name to text.
Common APIs	`gss_delete_sec_context`	Delete an established security context.
Common APIs	`gss_display_status`	Obtain the text error message associated with a GSS-API status code.
Common APIs	`gss_release_buffer`	Delete a buffer.
Common APIs	`gss_release_cred`	Release local data structures associated with a GSS-API credential.
Common APIs	`gss_release_name`	Delete internal format name.
Required definitions	`GSS_C_DELEG_FLAG`	Requests delegation.
Required definitions	`GSS_C_EMPTY_BUFFER`	Signifies that the `gss_buffer_desc` does not contain any data.
Required definitions	`GSS_C_GSS_CODE`	Indicates a GSS major status code.
Required definitions	`GSS_C_INDEFINITE`	Indicates that the mechanism does not support context expiration.
Required definitions	`GSS_C_MECH_CODE`	Indicates a GSS minor status code.
Required definitions	`GSS_C_MUTUAL_FLAG`	Mutual authentication requested.
Required definitions	`GSS_C_NO_BUFFER`	Signifies that the `gss_buffer_t` variable does not point to a valid `gss_buffer_desc` structure.
Required definitions	`GSS_C_NO_CHANNEL_BINDINGS`	No communication channel bindings.
Required definitions	`GSS_C_NO_CONTEXT`	Signifies that the `gss_ctx_id_t` variable does not point to a valid context.

Table 162. Required APIs and Definitions for GSS-API authentication plug-ins (continued)

Name		Description
Required definitions	`GSS_C_NO_CREDENTIAL`	Signifies that `gss_cred_id_t` variable does not point to a valid credential handle.
Required definitions	`GSS_C_NO_NAME`	Signifies that the `gss_name_t` variable does not point to a valid internal name.
Required definitions	`GSS_C_NO_OID`	Use default authentication mechanism.
Required definitions	`GSS_C_NULL_OID_SET`	Use default mechanism.
Required definitions	`GSS_S_COMPLETE`	API completed successfully.
Required definitions	`GSS_S_CONTINUE_NEEDED`	Processing is not complete and the API must be called again with the reply token received from the peer.

Related concepts:
- “Security plug-in APIs” on page 1425
- “Security plug-ins” on page 1397

Related reference:
- “Restrictions for GSS-API authentication plug-ins” on page 1461

Restrictions for GSS-API authentication plug-ins

The following is a list of restrictions for GSS-API authentication plug-ins.
- The default security mechanism is always assumed; therefore, there is no OID consideration.
- The only GSS services requested in `gss_init_sec_context()` are mutual authentication and delegation. DB2 always requests a ticket for delegation, but does not use that ticket to generate a new ticket.
- Only the default context time is requested.
- Context tokens from `gss_delete_sec_context()` are not sent from the client to the server and vice-versa.
- Anonymity is not supported.
- Channel binding is not supported
- If the initial credentials expire, DB2 does not automatically renew them.
- The GSS-API specification stipulates that even if `gss_init_sec_context()` or `gss_accept_sec_context()` fail, either function must return a token to send to the peer. However, because of DRDA limitations, DB2 only sends a token if `gss_init_sec_context()` fails and generates a token on the first call.

Related concepts:
- “Security plug-in APIs” on page 1425
- “Security plug-ins” on page 1397

Related reference:
- “Required APIs and definitions for GSS-API authentication plug-ins” on page 1460

Security plug-in API versioning

Because it is possible that future releases of DB2 will need different versions of the security plug-in APIs, DB2 supports version numbering of the APIs. These version numbers are integers starting with 1 for DB2 UDB version 8.2. The version number that DB2 passes to the security plug-in APIs are the highest version number of the API that DB2 can support, and correspond to the version number of the structure. If the plug-in can support a higher API version, it must return function pointers for the version that DB2 has requested. If the plug-in only supports a lower version of the API, the plug-in should fill in function pointers for the lower version. In either situation, the security plug-in APIs should return the version number for the API it is supporting in the version field of the functions structure.

For DB2, the version numbers of the security plug-ins will only change when necessary (for example, when there are changes to the parameters of the APIs). Version numbers will not automatically change with DB2 release numbers.

Related concepts:
- "Security plug-in APIs" on page 1425
- "Security plug-ins" on page 1397

Road map to the catalog views

Table 163. Road map to the read-only catalog views

Description	Catalog View
attributes of structured data types	
authorities on database	"SYSCAT.DBAUTH " on page 173
buffer pool configuration on database partition group	
buffer pool size on database partition	
cast functions	
check constraints	
column privileges	"SYSCAT.COLAUTH " on page 173
columns	
columns referenced by check constraints	
columns used in dimensions	
columns used in keys	
constraint dependencies	
database partition group database partitions	
database partition group definitions	
data partitions	
data types	
detailed column group statistics	
detailed column options	

Table 163. Road map to the read-only catalog views (continued)

Description	Catalog View
detailed column statistics	
distribution maps	
event monitor definitions	
events currently monitored	
function dependencies[1]	
function mapping	
function mapping options	
function parameter mapping options	
function parameters[1]	
functions[1]	
hierarchies (types, tables, views)	
identity columns	
index columns	
index dependencies	
index exploitation	
index extension dependencies	
index extension parameters	
index extension search methods	
index extensions	
index options	
index privileges	"SYSCAT.INDEXAUTH " on page 174
indexes	
method dependencies[1]	
method parameters[1]	
methods[1]	
nicknames	
object mapping	
package dependencies	"SYSCAT.PACKAGEDEP " on page 175
package privileges	"SYSCAT.PACKAGEAUTH " on page 175
packages	
partitioned tables	
pass-through privileges	"SYSCAT.PASSTHRUAUTH " on page 176
predicate specifications	
procedure options	
procedure parameter options	
procedure parameters[1]	
procedures[1]	

Table 163. Road map to the read-only catalog views (continued)

Description	Catalog View
protected tables	"SYSCAT.SECURITYLABELACCESS " on page 179
	"SYSCAT.SECURITYLABELCOMPONENTELEMENTS " on page 180
	"SYSCAT.SECURITYLABELCOMPONENTS " on page 180
	"SYSCAT.SECURITYLABELS " on page 180
	"SYSCAT.SECURITYPOLICIES " on page 181
	"SYSCAT.SECURITYPOLICYCOMPONENTRULES " on page 181
	"SYSCAT.SECURITYPOLICYEXEMPTIONS " on page 182
	"SYSCAT.SURROGATEAUTHIDS " on page 182
provides DB2 Universal Database for z/OS and OS/390 compatibility	
referential constraints	
remote table options	
routine dependencies	
routine parameters[1]	
routine privileges	"SYSCAT.ROUTINEAUTH " on page 177
routines[1]	
schema privileges	"SYSCAT.SCHEMAAUTH " on page 177
schemas	"SYSCAT.SCHEMATA " on page 178
sequence privileges	"SYSCAT.SEQUENCEAUTH " on page 179
sequences	"SYSCAT.SEQUENCES " on page 183
server options	
server options values	"SYSCAT.USEROPTIONS " on page 192
statements in packages	
stored procedures	
system servers	
table constraints	"SYSCAT.TABCONST " on page 184
table dependencies	
table privileges	"SYSCAT.TABAUTH " on page 193
table space use privileges	"SYSCAT.TBSPACEAUTH " on page 192
table spaces	"SYSCAT.TABLESPACES " on page 191
tables	"SYSCAT.TABLES " on page 185
transforms	
trigger dependencies	
triggers	
type mapping	
user-defined functions	
view dependencies	

Table 163. Road map to the read-only catalog views (continued)

Description	Catalog View
views	“SYSCAT.TABLES ” on page 185
wrapper options	
wrappers	
XML values index	
XSR objects	

[1] The catalog views for functions, methods, and procedures from DB2 Version 7.1 and earlier still exist. These views, however, do not reflect any changes since DB2 Version 7.1. The views are:

```
Functions:  SYSCAT.FUNCTIONS, SYSCAT.FUNCDEP, SYSCAT.FUNCPARMS
Methods:    SYSCAT.FUNCTIONS, SYSCAT.FUNCDEP, SYSCAT.FUNCPARMS
Procedures: SYSCAT.PROCEDURES, SYSCAT.PROCPARMS
```

Table 164. Road map to the updatable catalog views

Description	Catalog View
columns	
detailed column group statistics	
detailed column statistics	
indexes	
routines[1]	
tables	

[1] The SYSSTAT.FUNCTIONS catalog view still exists for updating the statistics for functions and methods. This view, however, does not reflect any changes since DB2 Version 7.1.

Road map to the catalog views

Chapter 47. Security Plug-In Configuration Parameters

clnt_krb_plugin - Client Kerberos plug-in

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null or IBMkrb5 [any valid string]

This parameter specifies the name of the default Kerberos plug-in library to be used for client-side authentication and local authorization. By default, the value is null on UNIX-based systems, and `IBMkrb5` on Windows operating systems. This plug-in is used when the client is authenticated using KERBEROS authentication.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

clnt_pw_plugin - Client userid-password plug-in

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

This parameter specifies the name of the userid-password plug-in library to be used for client-side authentication and local authorization. By default, the value is null and the DB2-supplied userid-password plug-in library is used. The plug-in is used when the client is authenticated using CLIENT, SERVER, or SERVER_ENCRYPT authentication.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*

- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

group_plugin - Group plug-in

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

This parameter specifies the name of the group plug-in library. By default, this value is null, and DB2 uses the operating system group lookup. The plug-in will be used for all group lookups.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

local_gssplugin - GSS API plug-in used for local instance level authorization

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

This parameter specifies the name of the default GSS API plug-in library to be used for instance level local authorization when the value of the *authentication* database manager configuration parameter is set to GSSPLUGIN or GSS_SERVER_ENCRYPT.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

srvcon_auth - Authentication type for incoming connections at the server

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [CLIENT; SERVER; SERVER_ENCRYPT; KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

This parameter specifies how and where user authentication is to take place when handling incoming connections at the server; it is used to override the current authentication type. If a value is not specified, DB2 uses the value of the *authentication* database manager configuration parameter.

For a description of each authentication type, see "authentication - Authentication type " on page 1497.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

This parameter specifies the GSS API plug-in libraries that are supported by the database server. By default, the value is null. If the authentication type is GSSPLUGIN and this parameter is NULL, an error is returned. If the authentication type is KERBEROS and this parameter is NULL, the DB2-supplied kerberos module or library is used. This parameter is not used if another authentication type is used.

When the authentication type is KERBEROS and the value of this parameter is not NULL, the list must contain exactly one Kerberos plug-in, and that plug-in is used for authentication (all other GSS plug-ins in the list are ignored). If there is more than one Kerberos plug-in, an error is returned.

Each GSS API plug-in name must be separated by a comma (,) with no space either before or after the comma. Plug-in names should be listed in the order of preference. This parameter handles incoming connections at the server when the *srvcon_auth* parameter is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT, or when *srvcon_auth* is not specified, and authentication is specified as KERBEROS, KRB_SERVER_ENCRYPT, GSSPLUGIN or GSS_SERVER_ENCRYPT.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	Null [any valid string]

This parameter specifies the name of the default userid-password plug-in library to be used for server-side authentication. By default, the value is null and the DB2-supplied userid-password plug-in library is used.

The parameter handles incoming connections at the server when the *srvcon_auth* parameter is specified as SERVER or SERVER_ENCRYPT, or when *srvcon_auth* is not specified, and *authentication* is specified as CLIENT, SERVER, or SERVER_ENCRYPT.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

srv_plugin_mode - Server plug-in mode

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients

	• Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	UNFENCED

This parameter specifies whether plug-ins are to run in fenced mode or unfenced mode. Unfenced mode is the only supported mode.

Related reference:

- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Part 15. Configuration parameters

Chapter 48. Configuration Parameters

Configuration parameters

When a DB2 database instance or a database is created, a corresponding configuration file is created with default parameter values. You can modify these parameter values to improve performance and other characteristics of the instance or database.

Configuration files contain parameters that define values such as the resources allocated to the DB2 database products and to individual databases, and the diagnostic level. There are two types of configuration files:

- The database manager configuration file for each DB2 instance
- The database configuration file for each individual database.

The *database manager configuration file* is created when a DB2 instance is created. The parameters it contains affect system resources at the instance level, independent of any one database that is part of that instance. Values for many of these parameters can be changed from the system default values to improve performance or increase capacity, depending on your system's configuration.

There is one database manager configuration file for each client installation as well. This file contains information about the client enabler for a specific workstation. A subset of the parameters available for a server are applicable to the client.

Database manager configuration parameters are stored in a file named db2systm. This file is created when the instance of the database manager is created. In UNIX-based environments, this file can be found in the sqllib subdirectory for the instance of the database manager. In Windows, the default location of this file is the instance subdirectory of the sqllib directory. If the DB2INSTPROF variable is set, the file is in the instance subdirectory of the directory specified by the DB2INSTPROF variable.

In a partitioned database environment, this file resides on a shared file system so that all database partition servers have access to the same file. The configuration of the database manager is the same on all database partition servers.

Most of the parameters either affect the amount of system resources that will be allocated to a single instance of the database manager, or they configure the setup of the database manager and the different communications subsystems based on environmental considerations. In addition, there are other parameters that serve informative purposes only and cannot be changed. All of these parameters have global applicability independent of any single database stored under that instance of the database manager.

A *database configuration file* is created when a database is created, and resides where that database resides. There is one configuration file per database. Its parameters specify, among other things, the amount of resource to be allocated to that database. Values for many of the parameters can be changed to improve performance or increase capacity. Different changes may be required, depending on the type of activity in a specific database.

Parameters for an individual database are stored in a configuration file named SQLDBCON. This file is stored along with other control files for the database in the SQLnnnnn directory, where nnnnn is a number assigned when the database was created. Each database has its own configuration file, and most of the parameters in the file specify the amount of resources allocated to that database. The file also contains descriptive information, as well as flags that indicate the status of the database.

In a partitioned database environment, a separate SQLDBCON file exists for each database partition. The values in the SQLDBCON file may be the same or different at each database partition, but the recommendation is that the database configuration parameter values be the same on all database partitions.

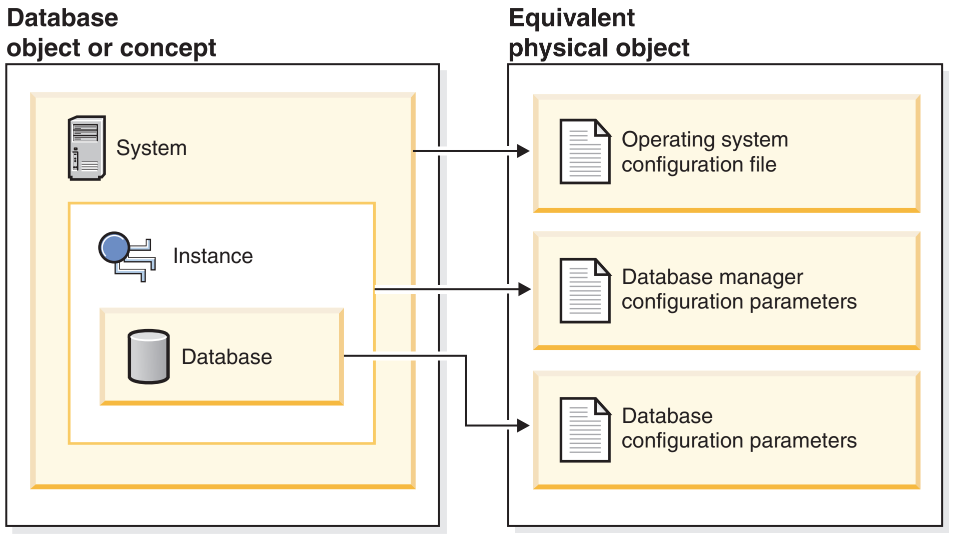


Figure 38. Relationship between database objects and configuration files

Related concepts:
- "Configuration parameters that affect query optimization" in *Performance Guide*

Related tasks:
- "Configuring DB2 with configuration parameters" on page 1493

Configuring DB2 with configuration parameters

The disk space and memory allocated by the database manager on the basis of default values of the parameters may be sufficient to meet your needs. In some situations, however, you may not be able to achieve maximum performance using these default values.

Since the default values are oriented towards machines that have relatively small memory resources and are dedicated as database servers, you may need to modify these values if your environment has:
- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

A good starting point for tuning your configuration is to use the Configuration Advisor or the AUTOCONFIGURE command which will generate values for parameters based on your responses to questions about workload characteristics.

Some configuration parameters can be set to *automatic*. DB2 will then automatically adjust these parameters to reflect the current resource requirements.

Database manager configuration parameters are stored in a file named `db2systm`. Database configuration parameters are stored in a file named `SQLDBCON`. These files cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.

Attention: If you edit `db2systm` or `SQLDBCON` using a method other than those provided by DB2, you may make the database unusable. **We strongly recommend** that you do not change these files using methods other than those documented and supported by DB2.

You may use one of the following methods to reset, update, and view configuration parameters:

- Using the Control Center. The Configure Instance notebook can be used to set the database manager configuration parameters on either a client or a server. The Configure Database notebook can be used to alter the value of database configuration parameters. The DB2 Control Center also provides the Configuration Advisor to alter the value of configuration parameters. This advisor generates values for parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database.

  In a partitioned database environment, the `SQLDBCON` file exists for each database partition. The Configure Database notebook will change the value on all database partitions if you launch the notebook from the database object in the tree view of the Control Center. If you launch the notebook from a database partition object, then it will only change the values for that database partition. (We recommend, however, that the configuration parameter values be the same on all database partitions.)

  Note: The Configuration Advisor is not available in the partitioned database environment.
- Using the command line processor. Commands to change the settings can be quickly and conveniently entered:

  For database manager configuration parameters:
  - GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)
  - UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
  - RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG) to reset *all* database manager parameters to their default values
  - AUTOCONFIGURE.

  For database configuration parameters:
  - GET DATABASE CONFIGURATION (or GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG) to reset *all* database parameters to their default values
  - AUTOCONFIGURE.
- Using application programming interfaces (APIs). The APIs can be called from an application or a host-language program. Call the following DB2 APIs to view or update configuration parameters:
  - db2AutoConfig - Access the Configuration Advisor
  - db2CfgGet - Get the database manager or database configuration parameters
  - db2CfgSet - Set the database manager or database configuration parameters

- Using the Configuration Assistant (for database manager configuration parameters). You can only use the Configuration Assistant to set the database manager configuration parameters on a client.

For some database manager configuration parameters, the database manager must be stopped (`db2stop`) and then restarted (`db2start`) for the new parameter values to take effect.

For some database parameters, changes will only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes will take effect.

Other parameters can be changed online; these are called *configurable online configuration parameters*.

If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command will be to apply the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

To change a database manager configuration parameter online:

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

For clients, changes to the database manager configuration parameters take effect the next time the client connects to a server.

If you change a configurable online database configuration parameter while connected, the default behavior is to apply the change online, wherever possible. You should note that some parameter changes may take a noticeable amount of time to take effect due to the overhead associated with allocating space. To change configuration parameters online from the command line processor, a connection to the database is required. To change a database configuration parameter online:

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

Each configurable online configuration parameter has a *propagation class* associated with it. The propagation class indicates when you can expect a change to the configuration parameter to take effect. There are three propagation classes:

- **Immediate**: Parameters that change immediately upon command or API invocation. For example, *diaglevel* has a propagation class of immediate.
- **Statement boundary**: Parameters that change on statement and statement-like boundaries. For example, if you change the value of *sortheap*, all new requests will start using the new value.
- **Transaction boundary**: Parameters that change on transaction boundaries. For example, a new value for *dl_expint* is updated after a COMMIT statement.

Changing some database configuration parameters can influence the access plan chosen by the SQL and XQuery optimizer. After changing any of these parameters, you should consider rebinding your applications to ensure the best access plan is being used for your SQL and XQuery statements. Any parameters that were modified online (for example, by using the UPDATE DATABASE

CONFIGURATION IMMEDIATE command) will cause the SQL and XQuery optimizer to choose new access plans for new query statements. However, the query statement cache will not be purged of existing entries. To clear the contents of the query cache, use the FLUSH PACKAGE CACHE statement.

While new parameter values may not be immediately effective, viewing the parameter settings (using GET DATABASE MANAGER CONFIGURATION or GET DATABASE CONFIGURATION commands) will always show the latest updates. Viewing the parameter settings using the SHOW DETAIL clause on these commands will show both the latest updates and the values in memory.

Note: A number of configuration parameters (for example, *userexit*) are described as having acceptable values of either “Yes” or “No”, or “On” or “Off” in the help and other DB2 documentation. To clarify, “Yes” should be considered equivalent to “On” and “No” should be considered equivalent to “Off”.

Related concepts:

- “Configuration parameters” on page 1475

Related reference:

- “Configuration parameters summary” on page 1484
- “FLUSH PACKAGE CACHE statement” in *SQL Reference, Volume 2*
- “AUTOCONFIGURE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “RESET DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2AutoConfig API - Access the Configuration Advisor” in *Administrative API Reference*
- “db2CfgGet - Get the database manager or database configuration parameters” on page 672
- “db2CfgSet - Set the database manager or database configuration parameters” on page 676
- “AUTOCONFIGURE command” in *Command Reference*
- “GET DATABASE CONFIGURATION ” on page 502
- “GET DATABASE MANAGER CONFIGURATION ” on page 508
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION ” on page 627
- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629

Configuring parameters dynamically

DB2 allows you to take advantage of dynamic configuration. You can change certain configuration parameters in a DB2 database or instance while that database is running, accepting connections or processing transactions. Following is an example of how you might take advantage of dynamic configuration features in DB2.

In this scenario, a database server is configured with 4 GB of memory, of which 3.5 GB are available to the database manager. There are 8 processors. The database server is dedicated to a single database called DB2DYN running on instance DB2INST. The database workload varies throughout the day and week, as follows:

- The daytime workload (05:00-20:00) includes many connections and concurrent transactions.
- The end of day workload (20:00-24:00) includes summary reports and decision-support queries, with few connections and transactions.
- The daily maintenance workload (24:00-05:00) includes online load operations, incremental backups, index creations, and so on.
- The weekly maintenance workload includes large table reorganization operations, runstats operations, and larger index creations.

Given these workload characteristics, you could configure the system as shown in the following table.

	Daytime (05:00 - 20:00)	**End of Day (20:00 - 24:00)**	**Daily Maintenance (24:00 - 05:00)**	**Weekly Maintenance (Sundays)**
Database activity	Heavy transaction workload	Decision-support queries	Load, backup, index creation	Reorg and runstats executed in buffer pool 1
Buffer pool 1 (MB)	1000	500	500	2000
Buffer pool 2 (MB)	1000	500	500	200
Sort heap (MB)	0.1	20	200	200
Catalog cache (MB)	200	200	50	50
Package cache (MB)	800	200	200	200
Utility heap (MB)	0	0	1000	0
Diag level	1	3	4	4

You could begin by setting the database configuration parameter *database_memory* to 3.5 GB, thereby reserving this specified amount of available memory for the database. This is a one time operation which, when complete, will give the database 3.5 gigabytes (or 917 504 4-kilobyte pages) of reserved memory for buffer pool creation or configuration adjustment.

```
db2start
db2 update db cfg for db2dyn using database_memory 917504
db2stop
```

You could then use the following scripts to transition the database from one configuration to another. (These scripts can be scheduled to run at the appropriate times.)

```
MorningConfiguration.sh

# This script is used to prepare
# the database server for the
# morning configuration,
# for a workload consisting of
# a large number of OLTP connections and
# concurent transactions.

db2 connect to db2dyn

db2 update db cfg using sortheap 25
db2 update db cfg using util_heap_sz 32
db2 alter bufferpool bufferpool1 size 262144
db2 commit
db2 update db cfg using catcachesz 51200
db2 update db cfg using pkcachesz 204800
db2 alter bufferpool bufferpool2 size 262144
db2 commit
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 1
db2 get dbm cfg show detail
db2 detach


EveningConfiguration.sh

# This script is used to prepare
# the database server for the
# evening configuration,
# for a workload consisting of
# decision-support queries.

db2 connect to db2dyn

db2 alter bufferpool bufferpool1 size 131072
db2 commit
db2 alter bufferpool bufferpool2 size 131072
db2 commit
db2 update db cfg using pkcachesz 51200
db2 update db cfg using catcachesz 51200
db2 update db cfg using util_heap_sz 32
db2 update db cfg using sortheap 5120
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 3
db2 get dbm cfg show detail
db2 detach


NightTimeConfiguration.sh

# This script is used to prepare
# the database server for the
# daily maintenance configuration,
# for a workload consisting of
# index creation and load and backup
# operations.

db2 connect to db2dyn

db2 alter bufferpool bufferpool1 size 131072
db2 commit
db2 alter bufferpool bufferpool2 size 131072
db2 commit
db2 update db cfg using catcachesz 51200
```

```
db2 update db cfg using pkcachesz 51200
db2 update db cfg using sortheap 51200
db2 update db cfg using util_heap_sz 262144
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 4
db2 get dbm cfg show detail
db2 detach


MaintenanceConfiguration.sh

# This script is used to prepare
# the database server for the
# weekly maintenance configuration,
# for a workload consisting of
# reorg and runstats operations executed
# in buffer pool 1.

db2 connect to db2dyn

db2 update db cfg using util_heap_sz 32
db2 alter bufferpool bufferpool2 size 51200
db2 commit
db2 update db cfg using sortheap 5120
db2 update db cfg using catcachesz 51200
db2 update db cfg using pkcachesz 51200
db2 alter bufferpool bufferpool1 size 524288
db2 commit
db2 flush package cache dynamic
db2 get db cfg show detail
db2 connect reset

db2 attach to instance db2inst
db2 update dbm cfg using diaglevel 4
db2 get dbm cfg show detail
db2 detach
```

Note that the order of operations differs from one script to the next. Operations that reduce memory requirements should be completed before operations that increase memory requirements; otherwise, the increases might fail.

All buffer pool resizings are followed by a commit operation, because buffer pool changes are SQL operations and are part of the transaction.

The package cache is flushed after each reconfiguration to signal the optimizer that the configuration has undergone a significant change, and that all existing dynamic SQL and XQuery access plans are now invalid.

Dynamic database manager reconfiguration operations are performed while an application is attached to the instance, and dynamic database operations are performed while an application is connected to the database.

The SHOW DETAIL clause on the GET DATABASE MANAGER CONFIGURATION command, or the GET DATABASE CONFIGURATION command, can be used to verify that a dynamic reconfiguration operation has taken effect.

Other database configuration parameters that could have been changed dynamically in this scenario include:

- The *locklist* parameter, which can be updated dynamically if the database experiences frequent lock escalations.
- The *dft_queryopt* parameter, which applies to new connections only, and can be used to increase the optimization level prior to the start of the decision-support workload.
- The *dft_degree* parameter, which also applies to new connections only, and can be used to provide decision-support queries with added intra-query parallelism.

Another option for addressing changing workload characteristics is to enable the self tuning memory feature, which distributes memory among different memory areas in response to changes in workload requirements.

Related concepts:

- "Self tuning memory" in *Performance Guide*

Configuration parameters summary

Database Manager Configuration Parameter Summary

The following table lists the parameters in the database manager configuration file for database servers. When changing the database manager configuration parameters, consider the detailed information for each parameter. Specific operating environment information including defaults is part of each parameter description.

For some database manager configuration parameters, the database manager must be stopped (`db2stop`) and restarted (`db2start`) for the new parameter values to take effect. Other parameters can be changed online; these are called *configurable online configuration parameters*. If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the UPDATE DBM CFG command applies the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

The column "Auto." in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DATABASE MANAGER CONFIGURATION command. If you set a parameter to automatic, DB2 will automatically adjust the parameter to reflect current resource requirements.

The column "Perf. Impact" provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High —** indicates the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium —** indicates the the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low —** indicates that the parameter has a less general or less significant impact on performance.

- **None —** indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns "Token", "Token Value", and "Data Type" provide information that you will need when calling the `db2CfgGet` or the `db2CfgSet` API. This information includes configuration parameter identifiers, entries for the *token* element in the *db2CfgParam* data structure, and data types for values that are passed to the structure.

Table 165. Configurable Database Manager Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
agent_stack_sz	No	No	Low	SQLF_KTN_AGENT_STACK_SZ	61	Uint16	
agentpri	No	No	High	SQLF_KTN_AGENTPRI	26	Sint16	
aslheapsz	No	No	High	SQLF_KTN_ASLHEAPSZ	15	Uint32	
audit_buf_sz	No	No	High	SQLF_KTN_AUDIT_BUF_SZ	312	Sint32	"audit_buf_sz - Audit buffer size " on page 1496
authentication[1]	No	No	Low	SQLF_KTN_AUTHENTICATION	78	Uint16	"authentication - Authentication type " on page 1497
catalog_noauth	Yes	No	None	SQLF_KTN_CATALOG_NOAUTH	314	Uint16	"catalog_noauth - Cataloging allowed without authority " on page 1499
clnt_krb_plugin	No	No	None	SQLF_KTN_CLNT_KRB_PLUGIN	812	char(33)	"clnt_krb_plugin - Client Kerberos plug-in " on page 1467
clnt_pw_plugin	No	No	None	SQLF_KTN_CLNT_PW_PLUGIN	811	char(33)	"clnt_pw_plugin - Client userid-password plug-in " on page 1467
comm_bandwidth	Yes	No	Medium	SQLF_KTN_COMM_BANDWIDTH	307	float	
conn_elapse	Yes	No	Medium	SQLF_KTN_CONN_ELAPSE	508	Uint16	"conn_elapse - Connection elapse time " on page 1513
cpuspeed	Yes	No	Low[2]	SQLF_KTN_CPUSPEED	42	float	
dft_account_str	Yes	No	None	SQLF_KTN_DFT_ACCOUNT_STR	28	char(25)	
dft_monswitches • *dft_mon_bufpool* • *dft_mon_lock* • *dft_mon_sort* • *dft_mon_stmt* • *dft_mon_table* • *dft_mon_timestamp* • *dft_mon_uow*	Yes	No	Medium	SQLF_KTN_DFT_MONSWITCHES[3] • SQLF_KTN_DFT_MON_BUFPOOL • SQLF_KTN_DFT_MON_LOCK • SQLF_KTN_DFT_MON_SORT • SQLF_KTN_DFT_MON_STMT • SQLF_KTN_DFT_MON_TABLE • SQLF_KTN_DFT_MON_ TIMESTAMP • SQLF_KTN_DFT_MON_UOW	29 • 33 • 34 • 35 • 31 • 32 • 36 • 30	Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16 • Uint16	
dftdbpath	Yes	No	None	SQLF_KTN_DFTDBPATH	27	char(215)	"dftdbpath - Default database path " on page 1500
diaglevel	Yes	No	Low	SQLF_KTN_DIAGLEVEL	64	Uint16	
diagpath	Yes	No	None	SQLF_KTN_DIAGPATH	65	char(215)	
dir_cache	No	No	Medium	SQLF_KTN_DIR_CACHE	40	Uint16	
discover[4]	No	No	Medium	SQLF_KTN_DISCOVER	304	Uint16	
discover_inst	Yes	No	Low	SQLF_KTN_DISCOVER_INST	308	Uint16	
fcm_num_buffers	Yes	Yes	Medium	SQLF_KTN_FCM_NUM_BUFFERS	503	Uint32	"fcm_num_buffers - Number of FCM buffers " on page 1514
fcm_num_channels	Yes	Yes	Medium	SQLF_KTN_FCM_NUM_CHANNELS	902	Uint32	"fcm_num_channels - Number of FCM channels " on page 1515
fed_noauth	Yes	No	None	SQLF_KTN_FED_NOAUTH	806	Uint16	
federated	No	No	Medium	SQLF_KTN_FEDERATED	604	Sint16	
fenced_pool	No	No	Medium	SQLF_KTN_FENCED_POOL	80	Sint32	

Table 165. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
group_plugin	No	No	None	SQLF_KTN_GROUP_PLUGIN	810	char(33)	"group_plugin - Group plug-in " on page 1468
health_mon	Yes	No	Low	SQLF_KTN_HEALTH_MON	804	Uint16	
*indexrec*5	Yes	No	Medium	SQLF_KTN_INDEXREC	20	Uint16	
instance_memory	No	Yes	Medium	SQLF_KTN_INSTANCE_MEMORY	803	Uint64	
intra_parallel	No	No	High	SQLF_KTN_INTRA_PARALLEL	306	Sint16	
java_heap_sz	No	No	High	SQLF_KTN_JAVA_HEAP_SZ	310	Sint32	
jdk_path	No	No	None	SQLF_KTN_JDK_PATH	311	char(255)	
keepfenced	No	No	Medium	SQLF_KTN_KEEPFENCED	81	Uint16	
local_gssplugin	No	No	None	SQLF_KTN_LOCAL_GSSPLUGIN	816	char(33)	"local_gssplugin - GSS API plug-in used for local instance level authorization " on page 1468
max_connections	No	No	Medium	SQLF_DBTN_MAX_CONNECTIONS	802	Sint32	
max_connretries	Yes	No	Medium	SQLF_KTN_MAX_CONNRETRIES	509	Uint16	"max_connretries - Node connection retries " on page 1516
max_coordagents	No	No	Medium	SQLF_KTN_MAX_COORDAGENTS	501	Sint32	
max_querydegree	Yes	No	High	SQLF_KTN_MAX_QUERYDEGREE	303	Sint32	
max_time_diff	No	No	Medium	SQLF_KTN_MAX_TIME_DIFF	510	Uint16	"max_time_diff - Maximum time difference among nodes " on page 1516
maxagents	No	No	Medium	SQLF_KTN_MAXAGENTS	12	Uint32	
maxcagents	No	No	Medium	SQLF_KTN_MAXCAGENTS	13	Sint32	
maxtotfilop	No	No	Medium	SQLF_KTN_MAXTOTFILOP	45	Uint16	
mon_heap_sz	No	No	Low	SQLF_KTN_MON_HEAP_SZ	79	Uint16	
nname	No	No	None	SQLF_KTN_NNAME	7	char(8)	
notifylevel	Yes	No	Low	SQLF_KTN_NOTIFYLEVEL	605	Sint16	
num_initagents	No	No	Medium	SQLF_KTN_NUM_INITAGENTS	500	Uint32	
num_initfenced	No	No	Medium	SQLF_KTN_NUM_INITFENCED	601	Sint32	
num_poolagents	No	No	High	SQLF_KTN_NUM_POOLAGENTS	502	Sint32	
numdb	No	No	Low	SQLF_KTN_NUMDB	6	Uint16	
query_heap_sz	No	No	Medium	SQLF_KTN_QUERY_HEAP_SZ	49	Sint32	
resync_interval	No	No	None	SQLF_KTN_RESYNC_INTERVAL	68	Uint16	
rqrioblk	No	No	High	SQLF_KTN_RQRIOBLK	1	Uint16	
sheapthres	No	No	High	SQLF_KTN_SHEAPTHRES	21	Uint32	
spm_log_file_sz	No	No	Low	SQLF_KTN_SPM_LOG_FILE_SZ	90	Sint32	
spm_log_path	No	No	Medium	SQLF_KTN_SPM_LOG_PATH	313	char(226)	
spm_max_resync	No	No	Low	SQLF_KTN_SPM_MAX_RESYNC	91	Sint32	
spm_name	No	No	None	SQLF_KTN_SPM_NAME	92	char(8)	
srvcon_auth	No	No	None	SQLF_KTN_SRVCON_AUTH	815	Uint16	"srvcon_auth - Authentication type for incoming connections at the server " on page 1469
srvcon_gssplugin_list	No	No	None	SQLF_KTN_SRVCON_GSSPLUGIN_ LIST	814	char(256)	"srvcon_gssplugin_list - List of GSS API plug-ins for incoming connections at the server " on page 1469
srv_plugin_mode	No	No	None	SQLF_KTN_SRV_PLUGIN_MODE	809	Uint16	"srv_plugin_mode - Server plug-in mode " on page 1470
srvcon_pw_plugin	No	No	None	SQLF_KTN_SRVCON_PW_PLUGIN	813	char(33)	"srvcon_pw_plugin - Userid-password plug-in for incoming connections at the server " on page 1470
start_stop_time	Yes	No	Low	SQLF_KTN_START_STOP_TIME	511	Uint16	"start_stop_time - Start and stop timeout " on page 1517
svcename	No	No	None	SQLF_KTN_SVCENAME	24	char(14)	"svcename - TCP/IP service name " on page 1501

Table 165. Configurable Database Manager Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
sysadm_group	No	No	None	SQLF_KTN_SYSADM_GROUP	39	char(30)	"sysadm_group - System administration authority group name " on page 1502
sysctrl_group	No	No	None	SQLF_KTN_SYSCTRL_GROUP	63	char(30)	"sysctrl_group - System control authority group name " on page 1503
sysmaint_group	No	No	None	SQLF_KTN_SYSMAINT_GROUP	62	char(30)	"sysmaint_group - System maintenance authority group name " on page 1503
sysmon_group	No	No	None	SQLF_KTN_SYSMON	808	char(30)	"sysmon_group - System monitor authority group name " on page 1504
tm_database	No	No	None	SQLF_KTN_TM_DATABASE	67	char(8)	
tp_mon_name	No	No	None	SQLF_KTN_TP_MON_NAME	66	char(19)	
trust_allclnts[6]	No	No	None	SQLF_KTN_TRUST_ALLCLNTS	301	Uint16	"trust_allclnts - Trust all clients " on page 1505
trust_clntauth	No	No	None	SQLF_KTN_TRUST_CLNTAUTH	302	Uint16	"trust_clntauth - Trusted clients authentication " on page 1506
util_impact_lim	Yes	No	High	SQLF_KTN_UTIL_IMPACT_LIM	807	Uint32	

Notes:

1. Valid values (defined in `sqlenv.h`):

```
SQL_AUTHENTICATION_SERVER (0)
SQL_AUTHENTICATION_CLIENT (1)
SQL_AUTHENTICATION_DCS (2)
SQL_AUTHENTICATION_DCE (3)
SQL_AUTHENTICATION_SVR_ENCRYPT (4)
SQL_AUTHENTICATION_DCS_ENCRYPT (5)
SQL_AUTHENTICATION_DCE_SVR_ENC (6)
SQL_AUTHENTICATION_KERBEROS (7)
SQL_AUTHENTICATION_KRB_SVR_ENC (8)
SQL_AUTHENTICATION_GSSPLUGIN (9)
SQL_AUTHENTICATION_GSS_SVR_ENC (10)
SQL_AUTHENTICATION_DATAENC (11)
SQL_AUTHENTICATION_DATAENC_CMP (12)
SQL_AUTHENTICATION_NOT_SPEC (255)
```

2. The *cpuspeed* parameter can have a significant impact on performance, but you should use the default value, except in very specific circumstances, as documented in the parameter description.
3.
```
Bit 1 (xxxx xxx1): dft_mon_uow
Bit 2 (xxxx xx1x): dft_mon_stmt
Bit 3 (xxxx x1xx): dft_mon_table
Bit 4 (xxxx 1xxx): dft_mon_buffpool
Bit 5 (xxx1 xxxx): dft_mon_lock
Bit 6 (xx1x xxxx): dft_mon_sort
Bit 7 (x1xx xxxx): dft_mon_timestamp
```

4. Valid values (defined in `sqlutil.h`):

```
SQLF_DSCVR_KNOWN (1)
SQLF_DSCVR_SEARCH (2)
```

5. Valid values (defined in `sqlutil.h`):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
```

6. Valid values (defined in `sqlutil.h`):

```
SQLF_TRUST_ALLCLNTS_NO (0)
SQLF_TRUST_ALLCLNTS_YES (1)
SQLF_TRUST_ALLCLNTS_DRDAONLY (2)
```

Table 166. Informational Database Manager Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
nodetype[1]	SQLF_KTN_NODETYPE	100	Uint16	"nodetype - Machine node type " on page 1519

Table 166. Informational Database Manager Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information
release	SQLF_KTN_RELEASE	101	Uint16	

Notes:

1. Valid values (defined in `sqlutil.h`):

```
SQLF_NT_STANDALONE (0)
SQLF_NT_SERVER (1)
SQLF_NT_REQUESTOR (2)
SQLF_NT_STAND_REQ (3)
SQLF_NT_MPP (4)
SQLF_NT_SATELLITE (5)
```

Database Configuration Parameter Summary

The following table lists the parameters in the database configuration file. When changing the database configuration parameters, consider the detailed information for the parameter.

For some database configuration parameters, changes only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) The changes take effect at the next connection to the database. Other parameters can be changed online; these are called *configurable online configuration parameters*.

The column “Auto.” in the following table indicates whether the parameter supports the AUTOMATIC keyword on the UPDATE DATABASE MANAGER CONFIGURATION command. If you set a parameter to automatic, DB2 will automatically adjust the parameter to reflect current resource requirements.

The column “Perf. Impact” provides an indication of the relative importance of each parameter as it relates to system performance. It is impossible for this column to apply accurately to all environments; you should view this information as a generalization.

- **High** — indicates that the parameter can have a significant impact on performance. You should consciously decide the values of these parameters, which, in some cases, means that you will accept the default values provided.
- **Medium** — indicates that the parameter can have some impact on performance. Your specific environment and needs will determine how much tuning effort should be focused on these parameters.
- **Low** — indicates that the parameter has a less general or less significant impact on performance.
- **None** — indicates that the parameter does not directly impact performance. Although you do not have to tune these parameters for performance enhancement, they can be very important for other aspects of your system configuration, such as communication support, for example.

The columns “Token”, “Token Value”, and “Data Type” provide information that you will need when calling the `db2CfgGet` or the `db2CfgSet` API. This information includes configuration parameter identifiers, entries for the *token* element in the *db2CfgParam* data structure, and data types for values that are passed to the structure.

Table 167. Configurable Database Configuration Parameters

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
alt_collate	No	No	None	SQLF_DBTN_ALT_COLLATE	809	Uint32	
app_ctl_heap_sz	No	No	Medium	SQLF_DBTN_APP_CTL_HEAP_SZ	500	Uint16	
appgroup_mem_sz	No	No	Medium	SQLF_DBTN_APPGROUP_MEM_SZ	800	Uint32	
applheapsz	No	No	Medium	SQLF_DBTN_APPLHEAPSZ	51	Uint16	
archretrydelay	Yes	No	None	SQLF_DBTN_ARCHRETRYDELAY	828	Uint16	
• *auto_maint* • *auto_db_backup* • *auto_tbl_maint* • *auto_runstats* • *auto_stats_prof* • *auto_prof_upd* • *auto_reorg*	Yes	No	Medium	• SQLF_ENABLE_AUTO_MAINT[1] • SQLF_ENABLE_AUTO_DB_BACKUP • SQLF_ENABLE_AUTO_TBL_MAINT • SQLF_ENABLE_AUTO_RUNSTATS • SQLF_ENABLE_AUTO_STATS_PROF • SQLF_ENABLE_AUTO_PROF_UPD • SQLF_ENABLE_AUTO_REORG	• 831 • 833 • 835 • 837 • 839 • 844 • 841	Uint32	
autorestart	Yes	No	Low	SQLF_DBTN_AUTO_RESTART	25	Uint16	"autorestart - Auto restart enable " on page 1518
avg_appls	Yes	Yes	High	SQLF_DBTN_AVG_APPLS	47	Uint16	
blk_log_dsk_ful	Yes	No	None	SQLF_DBTN_BLK_LOG_DSK_FUL	804	Uint16	
catalogcache_sz	Yes	No	High	SQLF_DBTN_CATALOGCACHE_SZ	56	Sint32	
chngpgs_thresh	No	No	High	SQLF_DBTN_CHNGPGS_THRESH	38	Uint16	
database_memory	No	Yes	Medium	SQLF_DBTN_DATABASE_MEMORY	803	Uint64	
dbheap	Yes	No	Medium	SQLF_DBTN_DB_HEAP	58	Uint64	
db_mem_thresh	Yes	No	High	SQLF_DBTN_DB_MEM_THRESH	849	Uint16	
dft_degree	Yes	No	High	SQLF_DBTN_DFT_DEGREE	301	Sint32	
dft_extent_sz	Yes	No	Medium	SQLF_DBTN_DFT_EXTENT_SZ	54	Uint32	
dft_loadrec_ses	Yes	No	Medium	SQLF_DBTN_DFT_LOADREC_SES	42	Sint16	
dft_mttb_types	No	No	None	SQLF_DBTN_DFT_MTTB_TYPES	843	Uint32	
dft_prefetch_sz	Yes	Yes	Medium	SQLF_DBTN_DFT_PREFETCH_SZ	40	Sint16	
dft_queryopt	Yes	No	Medium	SQLF_DBTN_DFT_QUERYOPT	57	Sint32	
dft_refresh_age	No	No	Medium	SQLF_DBTN_DFT_REFRESH_AGE	702	char(22)	
dft_sqlmathwarn	No	No	None	SQLF_DBTN_DFT_SQLMATHWARN	309	Sint16	
discover_db	Yes	No	Medium	SQLF_DBTN_DISCOVER	308	Uint16	
dlchktime	Yes	No	Medium	SQLF_DBTN_DLCHKTIME	9	Uint32	"dlchktime - Time interval for checking deadlock " on page 1507
dyn_query_mgmt	No	No	Low	SQLF_DBTN_DYN_QUERY_MGMT	604	Uint16	
failarchpath	Yes	No	None	SQLF_DBTN_FAILARCHPATH	826	char(243)	
groupheap_ratio	No	No	Medium	SQLF_DBTN_GROUPHEAP_RATIO	801	Uint16	
hadr_local_host	No	No	None	SQLF_DBTN_HADR_LOCAL_HOST	811	char(256)	
hadr_local_svc	No	No	None	SQLF_DBTN_HADR_LOCAL_SVC	812	char(41)	
hadr_remote_host	No	No	None	SQLF_DBTN_HADR_REMOTE_HOST	813	char(256)	
hadr_remote_inst	No	No	None	SQLF_DBTN_HADR_REMOTE_INST	815	char(9)	
hadr_remote_svc	No	No	None	SQLF_DBTN_HADR_REMOTE_SVC	814	char(41)	
hadr_syncmode	No	No	None	SQLF_DBTN_HADR_SYNCMODE	817	Uint32	
hadr_timeout	No	No	None	SQLF_DBTN_HADR_TIMEOUT	816	Sint32	
indexrec[2]	Yes	No	Medium	SQLF_DBTN_INDEXREC	30	Uint16	
locklist	Yes	Yes	High when it affects escalation	SQLF_DBTN_LOCK_LIST	704	Uint64	"locklist - Maximum storage for lock list " on page 1508

Table 167. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information
locktimeout	No	No	Medium	SQLF_DBTN_LOCKTIMEOUT	34	Sint16	"locktimeout - Lock timeout " on page 1510
logarchmeth1	Yes	No	None	SQLF_DBTN_LOGARCHMETH1	822	Uint16	
logarchmeth2	Yes	No	None	SQLF_DBTN_LOGARCHMETH2	823	Uint16	
logarchopt1	Yes	No	None	SQLF_DBTN_LOGARCHOPT1	824	char(243)	
logarchopt2	Yes	No	None	SQLF_DBTN_LOGARCHOPT2	825	char(243)	
logbufsz	No	No	High	SQLF_DBTN_LOGBUFSZ	33	Uint16	
logfilsiz	No	No	Medium	SQLF_DBTN_LOGFIL_SIZ	92	Uint32	
logindexbuild	Yes	Yes	None	SQLF_DBTN_LOGINDEXBUILD	818	Uint32	
logprimary	No	No	Medium	SQLF_DBTN_LOGPRIMARY	16	Uint16	
logretain[3]	No	No	Low	SQLF_DBTN_LOG_RETAIN	23	Uint16	
logsecond	Yes	No	Medium	SQLF_DBTN_LOGSECOND	17	Uint16	
max_log	Yes	Yes		SQLF_DBTN_MAX_LOG	807	Uint16	
maxappls	Yes	Yes	Medium	SQLF_DBTN_MAXAPPLS	6	Uint16	
maxfilop	Yes	No	Medium	SQLF_DBTN_MAXFILOP	3	Uint16	
maxlocks	Yes	Yes	High when it affects escalation	SQLF_DBTN_MAXLOCKS	15	Uint16	"maxlocks - Maximum percent of lock list before escalation " on page 1512
min_dec_div_3	No	No	High	SQLF_DBTN_MIN_DEC_DIV_3	605	Sint32	
mincommit	Yes	No	High	SQLF_DBTN_MINCOMMIT	32	Uint16	
mirrorlogpath	No	No	Low	SQLF_DBTN_MIRRORLOGPATH	806	char(242)	
newlogpath	No	No	Low	SQLF_DBTN_NEWLOGPATH	20	char(242)	
num_db_backups	Yes	No	None	SQLF_DBTN_NUM_DB_BACKUPS	601	Uint16	
num_freqvalues	Yes	No	Low	SQLF_DBTN_NUM_FREQVALUES	36	Uint16	
num_iocleaners	No	Yes	High	SQLF_DBTN_NUM_IOCLEANERS	37	Uint16	
num_ioservers	No	Yes	High	SQLF_DBTN_NUM_IOSERVERS	39	Uint16	
num_log_span	Yes	Yes		SQLF_DBTN_NUM_LOG_SPAN	808	Uint16	
num_quantiles	Yes	No	Low	SQLF_DBTN_NUM_QUANTILES	48	Uint16	
numarchretry	Yes	No	None	SQLF_DBTN_NUMARCHRETRY	827	Uint16	
overflowlogpath	No	No	Medium	SQLF_DBTN_OVERFLOWLOGPATH	805	char(242)	
pckcachesz	Yes	Yes	High	SQLF_DBTN_PCKCACHE_SZ	505	Uint32	
rec_his_retentn	No	No	None	SQLF_DBTN_REC_HIS_RETENTN	43	Sint16	
self_tuning_mem	Yes	No	High	SQLF_DBTN_SELF_TUNING_MEM	848	Uint16	
seqdetect	Yes	No	High	SQLF_DBTN_SEQDETECT	41	Uint16	
sheapthres_shr	No	Yes	High	SQLF_DBTN_SHEAPTHRES_SHR	802	Uint32	
softmax	No	No	Medium	SQLF_DBTN_SOFTMAX	5	Uint16	
sortheap	Yes	Yes	High	SQLF_DBTN_SORT_HEAP	52	Uint32	
stat_heap_sz	No	No	Low	SQLF_DBTN_STAT_HEAP_SZ	45	Uint32	
stmtheap	Yes	No	Medium	SQLF_DBTN_STMT_HEAP	821	Uint32	
trackmod	No	No	Low	SQLF_DBTN_TRACKMOD	703	Uint16	
tsm_mgmtclass	Yes	No	None	SQLF_DBTN_TSM_MGMTCLASS	307	char(30)	
tsm_nodename	Yes	No	None	SQLF_DBTN_TSM_NODENAME	306	char(64)	
tsm_owner	Yes	No	None	SQLF_DBTN_TSM_OWNER	305	char(64)	
tsm_password	Yes	No	None	SQLF_DBTN_TSM_PASSWORD	501	char(64)	
userexit	No	No	Low	SQLF_DBTN_USER_EXIT	24	Uint16	
util_heap_sz	Yes	No	Low	SQLF_DBTN_UTIL_HEAP_SZ	55	Uint32	
vendoropt	Yes	No	None	SQLF_DBTN_VENDOROPT	829	char(242)	

Table 167. Configurable Database Configuration Parameters (continued)

Parameter	Cfg. Online	Auto.	Perf. Impact	Token	Token Value	Data Type	Additional Information

Notes:

```
1. Default => Bit 1 on  (xxxx xxxx xxxx xxx1): auto_maint
              Bit 2 off (xxxx xxxx xxxx xx0x): auto_db_backup
              Bit 3 on  (xxxx xxxx xxxx x0xx): auto_tbl_maint
              Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
              Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
              Bit 6 off (xxxx xxxx xx0x xxxx): auto_prof_upd
              Bit 7 off (xxxx xxxx x0xx xxxx): auto_reorg
                           0    0     1   9

   Maximum => Bit 1 on  (xxxx xxxx xxxx xxx1): auto_maint
              Bit 2 off (xxxx xxxx xxxx xx1x): auto_db_backup
              Bit 3 on  (xxxx xxxx xxxx x1xx): auto_tbl_maint
              Bit 4 on  (xxxx xxxx xxxx 1xxx): auto_runstats
              Bit 5 off (xxxx xxxx xxx1 xxxx): auto_stats_prof
              Bit 6 off (xxxx xxxx xx1x xxxx): auto_prof_upd
              Bit 7 off (xxxx xxxx x1xx xxxx): auto_reorg
                           0    0    7    F
```

2. Valid values (defined in `sqlutil.h`):

```
SQLF_INX_REC_SYSTEM (0)
SQLF_INX_REC_REFERENCE (1)
SQLF_INX_REC_RESTART (2)
```

3. Valid values (defined in `sqlutil.h`):

```
SQLF_LOGRETAIN_NO (0)
SQLF_LOGRETAIN_RECOVERY (1)
SQLF_LOGRETAIN_CAPTURE (2)
```

Table 168. Informational Database Configuration Parameters

Parameter	Token	Token Value	Data Type	Additional Information
backup_pending	SQLF_DBTN_BACKUP_PENDING	112	Uint16	
codepage	SQLF_DBTN_CODEPAGE	101	Uint16	
codeset	SQLF_DBTN_CODESET	120	char(9)[1]	
collate_info	SQLF_DBTN_COLLATE_INFO	44	char(260)	
country/region	SQLF_DBTN_COUNTRY	100	Uint16	
database_consistent	SQLF_DBTN_CONSISTENT	111	Uint16	"database_consistent - Database is consistent " on page 1518
database_level	SQLF_DBTN_DATABASE_LEVEL	124	Uint16	
hadr_db_role	SQLF_DBTN_HADR_DB_ROLE	810	Uint32	
log_retain_status	SQLF_DBTN_LOG_RETAIN_STATUS	114	Uint16	
loghead	SQLF_DBTN_LOGHEAD	105	char(12)	
logpath	SQLF_DBTN_LOGPATH	103	char(242)	
multipage_alloc	SQLF_DBTN_MULTIPAGE_ALLOC	506	Uint16	
numsegs	SQLF_DBTN_NUMSEGS	122	Uint16	
pagesize	SQLF_DBTN_PAGESIZE	846	Uint32	
release	SQLF_DBTN_RELEASE	102	Uint16	
restore_pending	SQLF_DBTN_RESTORE_PENDING	503	Uint16	
restrict_access	SQLF_DBTN_RESTRICT_ACCESS	852	Sint32	"restrict_access - Database has restricted access configuration parameter" on page 1519

Table 168. Informational Database Configuration Parameters (continued)

Parameter	Token	Token Value	Data Type	Additional Information
rollfwd_pending	SQLF_DBTN_ROLLFWD_PENDING	113	Uint16	
territory	SQLF_DBTN_TERRITORY	121	char(5)[2]	
user_exit_status	SQLF_DBTN_USER_EXIT_STATUS	115	Uint16	

Notes:

1. char(17) on HP-UX and /.
2. char(33) on HP-UX and /.

DB2 Administration Server (DAS) Configuration Parameter Summary

Table 169. DAS Configuration Parameters

Parameter	Parameter Type	Additional Information
authentication	Configurable	"authentication - Authentication type DAS " on page 1498
contact_host	Configurable Online	
das_codepage	Configurable Online	
das_territory	Configurable Online	
dasadm_group	Configurable	"dasadm_group - DAS administration authority group name " on page 1500
db2system	Configurable Online	
discover	Configurable Online	
exec_exp_task	Configurable	
jdk_64_path	Configurable Online	
jdk_path	Configurable Online	
sched_enable	Configurable	
sched_userid	Informational	
smtp_server	Configurable Online	
toolscat_db	Configurable	
toolscat_inst	Configurable	
toolscat_schema	Configurable	

Changing the database configuration across multiple database partitions

Procedure:

When you have a database that is distributed across more than one database partition, the database configuration file should be the same on all database partitions. Consistency is required since the SQL compiler compiles distributed SQL statements based on information in the node configuration file and creates an access plan to satisfy the needs of the SQL statement. Maintaining different configuration files on database partitions could lead to different access plans, depending on which database partition the statement is prepared. Use `db2_all` to maintain the configuration files across all database partitions.

Related concepts:

- "Issuing commands in a partitioned database environment" on page 417

Related tasks:

- "Changing node and database configuration files" in *Administration Guide: Implementation*

Configuring DB2 with configuration parameters

The disk space and memory allocated by the database manager on the basis of default values of the parameters may be sufficient to meet your needs. In some situations, however, you may not be able to achieve maximum performance using these default values.

Since the default values are oriented towards machines that have relatively small memory resources and are dedicated as database servers, you may need to modify these values if your environment has:

- Large databases
- Large numbers of connections
- High performance requirements for a specific application
- Unique query or transaction loads or types

Each transaction processing environment is unique in one or more aspects. These differences can have a profound impact on the performance of the database manager when using the default configuration. For this reason, you are strongly advised to tune your configuration for your environment.

A good starting point for tuning your configuration is to use the Configuration Advisor or the AUTOCONFIGURE command which will generate values for parameters based on your responses to questions about workload characteristics.

Some configuration parameters can be set to *automatic*. DB2 will then automatically adjust these parameters to reflect the current resource requirements.

Database manager configuration parameters are stored in a file named `db2systm`. Database configuration parameters are stored in a file named `SQLDBCON`. These files cannot be directly edited, and can only be changed or viewed via a supplied API or by a tool which calls that API.

Attention: If you edit `db2systm` or `SQLDBCON` using a method other than those provided by DB2, you may make the database unusable. **We strongly recommend** that you do not change these files using methods other than those documented and supported by DB2.

You may use one of the following methods to reset, update, and view configuration parameters:

- Using the Control Center. The Configure Instance notebook can be used to set the database manager configuration parameters on either a client or a server. The Configure Database notebook can be used to alter the value of database configuration parameters. The DB2 Control Center also provides the Configuration Advisor to alter the value of configuration parameters. This advisor generates values for parameters based on the responses you provide to a set of questions, such as the workload and the type of transactions that run against the database.

In a partitioned database environment, the SQLDBCON file exists for each database partition. The Configure Database notebook will change the value on all database partitions if you launch the notebook from the database object in the tree view of the Control Center. If you launch the notebook from a database partition object, then it will only change the values for that database partition. (We recommend, however, that the configuration parameter values be the same on all database partitions.)

Note: The Configuration Advisor is not available in the partitioned database environment.

- Using the command line processor. Commands to change the settings can be quickly and conveniently entered:

  For database manager configuration parameters:
  - GET DATABASE MANAGER CONFIGURATION (or GET DBM CFG)
  - UPDATE DATABASE MANAGER CONFIGURATION (or UPDATE DBM CFG)
  - RESET DATABASE MANAGER CONFIGURATION (or RESET DBM CFG) to reset *all* database manager parameters to their default values
  - AUTOCONFIGURE.

  For database configuration parameters:
  - GET DATABASE CONFIGURATION (or GET DB CFG)
  - UPDATE DATABASE CONFIGURATION (or UPDATE DB CFG)
  - RESET DATABASE CONFIGURATION (or RESET DB CFG) to reset *all* database parameters to their default values
  - AUTOCONFIGURE.
- Using application programming interfaces (APIs). The APIs can be called from an application or a host-language program. Call the following DB2 APIs to view or update configuration parameters:
  - db2AutoConfig - Access the Configuration Advisor
  - db2CfgGet - Get the database manager or database configuration parameters
  - db2CfgSet - Set the database manager or database configuration parameters
- Using the Configuration Assistant (for database manager configuration parameters). You can only use the Configuration Assistant to set the database manager configuration parameters on a client.

For some database manager configuration parameters, the database manager must be stopped (`db2stop`) and then restarted (`db2start`) for the new parameter values to take effect.

For some database parameters, changes will only take effect when the database is reactivated. In these cases, all applications must first disconnect from the database. (If the database was activated, then it must be deactivated and reactivated.) Then, at the first new connect to the database, the changes will take effect.

Other parameters can be changed online; these are called *configurable online configuration parameters*.

If you change the setting of a configurable online database manager configuration parameter while you are attached to an instance, the default behavior of the

UPDATE DBM CFG command will be to apply the change immediately. If you do not want the change applied immediately, use the DEFERRED option on the UPDATE DBM CFG command.

To change a database manager configuration parameter online:

```
db2 attach to <instance-name>
db2 update dbm cfg using <parameter-name> <value>
db2 detach
```

For clients, changes to the database manager configuration parameters take effect the next time the client connects to a server.

If you change a configurable online database configuration parameter while connected, the default behavior is to apply the change online, wherever possible. You should note that some parameter changes may take a noticeable amount of time to take effect due to the overhead associated with allocating space. To change configuration parameters online from the command line processor, a connection to the database is required. To change a database configuration parameter online:

```
db2 connect to <dbname>
db2 update db cfg using <parameter-name> <parameter-value>
db2 connect reset
```

Each configurable online configuration parameter has a *propagation class* associated with it. The propagation class indicates when you can expect a change to the configuration parameter to take effect. There are three propagation classes:

- **Immediate**: Parameters that change immediately upon command or API invocation. For example, *diaglevel* has a propagation class of immediate.
- **Statement boundary**: Parameters that change on statement and statement-like boundaries. For example, if you change the value of *sortheap*, all new requests will start using the new value.
- **Transaction boundary**: Parameters that change on transaction boundaries. For example, a new value for *dl_expint* is updated after a COMMIT statement.

Changing some database configuration parameters can influence the access plan chosen by the SQL and XQuery optimizer. After changing any of these parameters, you should consider rebinding your applications to ensure the best access plan is being used for your SQL and XQuery statements. Any parameters that were modified online (for example, by using the UPDATE DATABASE CONFIGURATION IMMEDIATE command) will cause the SQL and XQuery optimizer to choose new access plans for new query statements. However, the query statement cache will not be purged of existing entries. To clear the contents of the query cache, use the FLUSH PACKAGE CACHE statement.

While new parameter values may not be immediately effective, viewing the parameter settings (using GET DATABASE MANAGER CONFIGURATION or GET DATABASE CONFIGURATION commands) will always show the latest updates. Viewing the parameter settings using the SHOW DETAIL clause on these commands will show both the latest updates and the values in memory.

Note: A number of configuration parameters (for example, *userexit*) are described as having acceptable values of either “Yes” or “No”, or “On” or “Off” in the help and other DB2 documentation. To clarify, “Yes” should be considered equivalent to “On” and “No” should be considered equivalent to “Off”.

Related concepts:

- “Configuration parameters” on page 1475

Related reference:

- “Configuration parameters summary” on page 1484
- “FLUSH PACKAGE CACHE statement” in *SQL Reference, Volume 2*
- “AUTOCONFIGURE command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “RESET DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “RESET DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “UPDATE DATABASE MANAGER CONFIGURATION command using the ADMIN_CMD procedure” in *Administrative SQL Routines and Views*
- “db2AutoConfig API - Access the Configuration Advisor” in *Administrative API Reference*
- “db2CfgGet - Get the database manager or database configuration parameters” on page 672
- “db2CfgSet - Set the database manager or database configuration parameters” on page 676
- “AUTOCONFIGURE command” in *Command Reference*
- “GET DATABASE CONFIGURATION ” on page 502
- “GET DATABASE MANAGER CONFIGURATION ” on page 508
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION ” on page 627
- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629

Security-Related Configuration Parameters

audit_buf_sz - Audit buffer size

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	0 [0 – 65 000]
Unit of Measure	Pages (4 KB)
When Allocated	When DB2 is started
When Freed	When DB2 is stopped

This parameter specifies the size of the buffer used when auditing the database.

The default value for this parameter is zero (0). If the value is zero (0), the audit buffer is not used. If the value is greater than zero (0), space is allocated for the audit buffer where the audit records will be placed when they are generated by the audit facility. The value times 4 KB pages is the amount of space allocated for the audit buffer. The audit buffer cannot be allocated dynamically; DB2 must be stopped and then restarted before the new value for this parameter takes effect.

By changing this parameter from the default to some value larger than zero (0), the audit facility writes records to disk asynchronously compared to the execution of the statements generating the audit records. This improves DB2 performance over leaving the parameter value at zero (0). The value of zero (0) means the audit facility writes records to disk synchronously with (at the same time as) the execution of the statements generating the audit records. The synchronous operation during auditing decreases the performance of applications running in DB2.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

authentication - Authentication type

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	SERVER [CLIENT; SERVER; SERVER_ENCRYPT; DATA_ENCRYPT; DATA_ENCRYPT_CMP KERBEROS; KRB_SERVER_ENCRYPT; GSSPLUGIN; GSS_SERVER_ENCRYPT]

This parameter specifies and determines how and where authentication of a user takes place.

If authentication is SERVER, the user ID and password are sent from the client to the server so that authentication can take place on the server. The value SERVER_ENCRYPT provides the same behavior as SERVER, except that any passwords sent over the network are encrypted.

A value of DATA_ENCRYPT means the server accepts encrypted SERVER authentication schemes and the encryption of user data. The authentication works exactly the same way as SERVER_ENCRYPT.

The following user data are encrypted when using this authentication type:
- SQL statements
- SQL program variable data

- Output data from the server processing an SQL statement and including a description of the data
- Some or all of the answer set data resulting from a query
- Large object (LOB) streaming
- SQLDA descriptors

A value of DATA_ENCRYPT_CMP means the server accepts encrypted SERVER authentication schemes and the encryption of user data. In addition, this authentication type allows compatibility with earlier products that do not support DATA_ENCRYPT authentication type. These products are permitted to connect with the SERVER_ENCRYPT authentication type and without encrypting user data. Products supporting the new authentication type must use it. This authentication type is only valid in the server's database manager configuration file and is not valid when used on the CATALOG DATABASE command.

Note: For a Common Criteria compliant configuration, SERVER is the only supported value.

A value of CLIENT indicates that all authentication takes place at the client. No authentication needs to be performed at the server.

A value of KERBEROS means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication. With an authentication type of KRB_SERVER_ENCRYPT at the server and clients that support the Kerberos security system, the effective system authentication type is KERBEROS. If the clients do not support the Kerberos security system, the system authentication type is effectively equivalent to SERVER_ENCRYPT.

A value of GSSPLUGIN means that authentication is performed using an external GSSAPI-based security mechanism. With an authentication type of GSS_SERVER_ENCRYPT at the server and clients that support the GSSPLUGIN security mechanism, the effective system authentication type is GSSPLUGIN (that is, if the clients support one of the server's plug-ins). If the clients do not support the GSSPLUGIN security mechanism, the system authentication type is effectively equivalent to SERVER_ENCRYPT.

Recommendation: Typically, the default value (SERVER) is adequate.

Related concepts:
- "Authentication methods for your server" on page 89

Related reference:
- "Restrictions on native XML data store" in *XML Guide*
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

authentication - Authentication type DAS

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable

Default [Range]	SERVER_ENCRYPT [SERVER_ENCRYPT; KERBEROS_ENCRYPT]

This parameter determines how and where authentication of a user takes place.

If authentication is SERVER_ENCRYPT, then the user ID and password are sent from the client to the server so authentication can take place on the server. Passwords sent over the network are encrypted.

A value of KERBEROS_ENCRYPT means that authentication is performed at a Kerberos server using the Kerberos security protocol for authentication.

Note: The KERBEROS_ENCRYPT authentication type is only supported on servers running Windows 2000.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

catalog_noauth - Cataloging allowed without authority

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	**Database server with local and remote clients** NO [NO (0) — YES (1)] **Client; Database server with local clients** YES [NO (0) — YES (1)]

This parameter specifies whether users are able to catalog and uncatalog databases and nodes, or DCS and ODBC directories, without SYSADM authority. The default value (0) for this parameter indicates that SYSADM authority is required. When this parameter is set to 1 (yes), SYSADM authority is not required.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

dasadm_group - DAS administration authority group name

Configuration Type	DB2 Administration Server
Applies to	DB2 Administration Server
Parameter Type	Configurable
Default [Range]	Null [any valid group name]

DAS Administration (DASADM) authority is the highest level of authority within the DAS. This parameter defines the group name with DASADM authority for the DAS.

DASADM authority is determined by the security facilities used in a specific operating environment.

- For the Windows operating systems, this parameter can be set to any local group that is defined in the Windows security database. Group names are accepted as long as they are 30 bytes or less in length.If "NULL" is specified for this parameter, all members of the Administrators group have DASADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the DASADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the DASADM group can be any valid UNIX group name.

This parameter can only be updated from a Version 8 command line processor (CLP).

Related reference:
- "GET ADMIN CONFIGURATION command" in *Command Reference*
- "RESET ADMIN CONFIGURATION command" in *Command Reference*
- "UPDATE ADMIN CONFIGURATION command" in *Command Reference*

dftdbpath - Default database path

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	**UNIX** Home directory of instance owner [any existing path] **Windows** Drive on which DB2 is installed [any existing path]

This parameter contains the default file path used to create databases under the database manager. If no path is specified when a database is created, the database is created under the path specified by the *dftdbpath* parameter.

In a partitioned database environment, you should ensure that the path on which the database is being created is not an NFS-mounted path (on UNIX-based platforms), or a network drive (in a Windows environment). The specified path must physically exist on each database partition server. To avoid confusion, it is best to specify a path that is locally mounted on each database partition server. The maximum length of the path is 205 characters. The system appends the node name to the end of the path.

Given that databases can grow to a large size and that many users could be creating databases (depending on your environment and intentions), it is often convenient to be able to have all databases created and stored in a specified location. It is also good to be able to isolate databases from other applications and data both for integrity reasons and for ease of backup and recovery.

For UNIX-based environments, the length of the *dftdbpath* name cannot exceed 215 characters and must be a valid, absolute, path name. For Windows, the *dftdbpath* can be a drive letter, optionally followed by a colon.

Recommendation: If possible, put high volume databases on a different disk than other frequently accessed data, such as the operating system files and the database logs.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

svcename - TCP/IP service name

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter contains the name of the TCP/IP port which a database server will use to await communications from remote client nodes. This name must be the first of two consecutive ports reserved for use by the database manager; the second port is used to handle interrupt requests from down-level clients.

In order to accept connection requests from a database client using TCP/IP, the database server must be listening on a port designated to that server. The system administrator for the database server must reserve a port (number *n*) and define its associated TCP/IP service name in the services file at the server. If the database server needs to support requests from down-level clients, a second port (number *n+1*, for interrupt requests) needs to be defined in the services file at the server.

The database server port (number *n*) and its TCP/IP service name need to be defined in the services file on the database client. Down-level clients also require the interrupt port (number *n+1*) to be defined in the client's services file.

On UNIX-based systems, the services file is located in: `/etc/services`

The *svcename* parameter should be set to the service name associated with the main connection port so that when the database server is started, it can determine on which port to listen for incoming connection requests. If you are supporting or using a down-level client, the service name for the interrupt port is not saved in the configuration file. The interrupt port number can be derived based on the main connection port number (*interrupt port number* = *main connection port* + 1).

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

sysadm_group - System administration authority group name

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

System administration (SYSADM) authority is the highest level of authority within the database manager and controls all database objects. This parameter defines the group name with SYSADM authority for the database manager instance.

SYSADM authority is determined by the security facilities used in a specific operating environment.
- For the Windows operating system, this parameter can be set to any local group, and is defined in the Windows 2000 security database. Group names must be 30 bytes or less in length. If "NULL" is specified for this parameter, all members of the Administrators group have SYSADM authority.
- For UNIX-based systems, if "NULL" is specified as the value of this parameter, the SYSADM group defaults to the primary group of the instance owner.

  If the value is not "NULL", the SYSADM group can be any valid UNIX group name.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSADM_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:

- “GET DATABASE MANAGER CONFIGURATION ” on page 508
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629
- “sysctrl_group - System control authority group name ” on page 1503
- “sysmaint_group - System maintenance authority group name ” on page 1503
- “sysmon_group - System monitor authority group name ” on page 1504

sysctrl_group - System control authority group name

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter defines the group name with system control (SYSCTRL) authority. SYSCTRL has privileges allowing operations affecting system resources, but does not allow direct access to data.

Group names on all platforms are accepted as long as they are 30 bytes or less in length.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSCTRL group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSCTRL_GROUP NULL. You must specify the keyword “NULL” in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:
- “GET DATABASE MANAGER CONFIGURATION ” on page 508
- “RESET DATABASE MANAGER CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE MANAGER CONFIGURATION ” on page 629
- “sysadm_group - System administration authority group name ” on page 1502
- “sysmaint_group - System maintenance authority group name ” on page 1503
- “sysmon_group - System monitor authority group name ” on page 1504

sysmaint_group - System maintenance authority group name

Configuration Type	Database manager

Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter defines the group name with system maintenance (SYSMAINT) authority. SYSMAINT has privileges to perform maintenance operations on all databases associated with an instance without having direct access to data.

Group names on all platforms are accepted as long as they are 30 bytes or less in length.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMAINT group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMAINT_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629
- "sysadm_group - System administration authority group name " on page 1502
- "sysctrl_group - System control authority group name " on page 1503
- "sysmon_group - System monitor authority group name " on page 1504

sysmon_group - System monitor authority group name

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default	Null

This parameter defines the group name with system monitor (SYSMON) authority. Users having SYSMON authority at the instance level have the ability to take database system monitor snapshots of a database manager instance or its databases. SYSMON authority includes the ability to use the following commands:
- GET DATABASE MANAGER MONITOR SWITCHES
- GET MONITOR SWITCHES
- GET SNAPSHOT
- LIST ACTIVE DATABASES
- LIST APPLICATIONS
- LIST DCS APPLICATIONS
- RESET MONITOR
- UPDATE MONITOR SWITCHES

Users with SYSADM, SYSCTRL, or SYSMAINT authority automatically have the ability to take database system monitor snapshots and to use these commands.

Group names on all platforms are accepted as long as they are 30 bytes or less in length.

Attention: This parameter must be NULL for Windows 98 clients when system security is used (that is, authentication is CLIENT, SERVER, DCS, or any other valid authentication). This is because the Windows 98 operating systems do not store group information, thereby providing no way of determining if a user is a member of a designated SYSMON group. When a group name is specified, no user can be a member of it.

To restore the parameter to its default (NULL) value, use UPDATE DBM CFG USING SYSMON_GROUP NULL. You must specify the keyword "NULL" in uppercase. You can also use the Configure Instance notebook in the DB2 Control Center.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629
- "sysadm_group - System administration authority group name " on page 1502
- "sysctrl_group - System control authority group name " on page 1503
- "sysmaint_group - System maintenance authority group name " on page 1503

trust_allclnts - Trust all clients

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	YES [NO, YES, DRDAONLY]

This parameter is only active when the *authentication* parameter is set to CLIENT.

This parameter and *trust_clntauth* are used to determine where users are validated to the database environment.

By accepting the default of "YES" for this parameter, all clients are treated as trusted clients. This means that the server assumes that a level of security is available at the client and the possibility that users can be validated at the client.

This parameter can only be changed to "NO" if the *authentication* parameter is set to CLIENT. If this parameter is set to "NO", the untrusted clients must provide a userid and password combination when they connect to the server. Untrusted clients are operating system platforms that do not have a security subsystem for authenticating users.

Setting this parameter to "DRDAONLY" protects against all clients except clients from DB2 for OS/390 and z/OS, DB2 for VM and VSE, and DB2 for OS/400®. Only these clients can be trusted to perform client-side authentication. All other clients must provide a user ID and password to be authenticated by the server.

When *trust_allclnts* is set to "DRDAONLY", the *trust_clntauth* parameter is used to determine where the clients are authenticated. If *trust_clntauth* is set to "CLIENT", authentication occurs at the client. If *trust_clntauth* is set to "SERVER", authentication occurs at the client if no password is provided, and at the server if a password is provided.

Related reference:
- "authentication - Authentication type " on page 1497
- "trust_clntauth - Trusted clients authentication " on page 1506
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

trust_clntauth - Trusted clients authentication

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	CLIENT [CLIENT, SERVER]

This parameter specifies whether a trusted client is authenticated at the server or the client when the client provides a userid and password combination for a connection. This parameter (and *trust_allclnts*) is only active if the *authentication* parameter is set to CLIENT. If a user ID and password are not provided, the client is assumed to have validated the user, and no further validation is performed at the server.

If this parameter is set to CLIENT (the default), the trusted client can connect without providing a user ID and password combination, and the assumption is that the operating system has already authenticated the user. If it is set to SERVER, the user ID and password will be validated at the server.

The numeric value for CLIENT is 0. The numeric value for SERVER is 1.

Related reference:
- "authentication - Authentication type " on page 1497
- "trust_allclnts - Trust all clients " on page 1505
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

Locking Configuration Parameters

dlchktime - Time interval for checking deadlock

Configuration Type	Database
Parameter Type	Configurable online
Propagation Class	Immediate
Default [Range]	10 000 (10 seconds) [1 000 – 600 000]
Unit of Measure	Milliseconds

A deadlock occurs when two or more applications connected to the same database wait indefinitely for a resource. The waiting is never resolved because each application is holding a resource that the other needs to continue.

The deadlock check interval defines the frequency at which the database manager checks for deadlocks among all the applications connected to a database.

Notes:
1. In a partitioned database environment, this parameter applies to the catalog node only.
2. In a partitioned database environment, a deadlock is not flagged until after the second iteration.

Recommendation: Increasing this parameter decreases the frequency of checking for deadlocks, thereby increasing the time that application programs must wait for the deadlock to be resolved.

Decreasing this parameter increases the frequency of checking for deadlocks, thereby decreasing the time that application programs must wait for the deadlock to be resolved but increasing the time that the database manager takes to check for deadlocks. If the deadlock interval is too small, it can decrease run-time performance, because the database manager is frequently performing deadlock detection. If this parameter is set lower to improve concurrency, you should ensure that *maxlocks* and *locklist* are set appropriately to avoid unnecessary lock escalation, which can result in more lock contention and as a result, more deadlock situations.

Related reference:

- “locklist - Maximum storage for lock list ” on page 1508
- “maxlocks - Maximum percent of lock list before escalation ” on page 1512
- “GET DATABASE CONFIGURATION ” on page 502
- “RESET DATABASE CONFIGURATION command” in *Command Reference*
- “UPDATE DATABASE CONFIGURATION ” on page 627

locklist - Maximum storage for lock list

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	
	UNIX Automatic [4 – 524 288]
	Windows Database server with local and remote clients Automatic [4 – 524 288]
	Windows 64-bit Database server with local clients Automatic [4 – 524 288]
	Windows 32-bit Database server with local clients Automatic [4 – 524 288]
Unit of Measure	Pages (4 KB)
When Allocated	When the first application connects to the database
When Freed	When last application disconnects from the database

This parameter indicates the amount of storage that is allocated to the lock list. There is one lock list per database and it contains the locks held by all applications concurrently connected to the database. Locking is the mechanism that the database manager uses to control concurrent access to data in the database by multiple applications. Both rows and tables can be locked. The database manager can also acquire locks for internal use.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active

The value of *locklist* is tuned together with the *maxlocks* parameter, therefore disabling self tuning of the *locklist* parameter automatically disables self tuning of the *maxlocks* parameter. Enabling self tuning of the *locklist* parameter automatically enables self tuning of the *maxlocks* parameter.

On 32-bit platforms, each lock requires 48 or 96 bytes of the lock list, depending on whether other locks are held on the object:

- 96 bytes are required to hold a lock on an object that has no other locks held on it
- 48 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit platforms (except HP-UX/PA-RISC), each lock requires 64 or 128 bytes of the lock list, depending on whether other locks are held on the object:

- 128 bytes are required to hold a lock on an object that has no other locks held on it
- 64 bytes are required to record a lock on an object that has an existing lock held on it.

On 64-bit HP-UX/PA-RISC, each lock requires 80 or 160 bytes of the lock list, depending on whether or not other locks are held on the object.

When the percentage of the lock list used by one application reaches *maxlocks*, the database manager will perform lock escalation, from row to table, for the locks held by the application. Although the escalation process itself does not take much time, locking entire tables (versus individual rows) decreases concurrency, and overall database performance might decrease for subsequent accesses against the affected tables. Suggestions of how to control the size of the lock list are:

- Perform frequent COMMITs to release locks.
- When performing many updates, lock the entire table before updating (using the SQL LOCK TABLE statement). This will use only one lock, keeps others from interfering with the updates, but does reduce concurrency of the data.

  You can also use the LOCKSIZE option of the ALTER TABLE statement to control how locking is done for a specific table.

  Use of the Repeatable Read isolation level might result in an automatic table lock.
- Use the Cursor Stability isolation level when possible to decrease the number of share locks held. If application integrity requirements are not compromised use Uncommitted Read instead of Cursor Stability to further decrease the amount of locking.
- Set *locklist* to AUTOMATIC. The lock list will increase synchronously to avoid lock escalation or a lock list full situation.

Once the lock list is full, performance can degrade since lock escalation will generate more table locks and fewer row locks, thus reducing concurrency on shared objects in the database. Additionally there might be more deadlocks between applications (since they are all waiting on a limited number of table locks), which will result in transactions being rolled back. Your application will receive an SQLCODE of -912 when the maximum number of lock requests has been reached for the database.

Recommendation: If lock escalations are causing performance concerns you might need to increase the value of this parameter or the *maxlocks* parameter. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *lock_escals (lock escalations)* monitor element.

The following steps might help in determining the number of pages required for your lock list:

1. Calculate a lower bound for the size of your lock list, using *one* of the following calculations, depending on your environment:
   a. `(512 * x * maxappls) / 4096`
   b. with Concentrator enabled:
      ```
      (512 * x * max_coordagents) / 4096
      ```
   c. in a partitioned database with Concentrator enabled:
      ```
      (512 * x * max_coordagents * number of database partitions) / 4096
      ```

where 512 is an estimate of the average number of locks per application and x is the number of bytes required for each lock against an object that has an existing lock (40 bytes on 32-bit platforms, 64 bytes on 64-bit platforms).

2. Calculate an upper bound for the size of your lock list:

   ```
   (512 * y * maxappls) / 4096
   ```

   where y is the number of bytes required for the first lock against an object (80 bytes on 32-bit platforms, 128 bytes on 64-bit platforms).
3. Estimate the amount of concurrency you will have against your data and based on your expectations, choose an initial value for *locklist* that falls between the upper and lower bounds that you have calculated.
4. Using the database system monitor, as described below, tune the value of this parameter.

You can use the database system monitor to determine the maximum number of locks held by a given transaction. Refer to the *locks_held_top (maximum number of locks held)* monitor element.

This information can help you validate or adjust the estimated number of locks per application. In order to perform this validation, you will have to sample several applications, noting that the monitor information is provided at a transaction level, not an application level.

You might also want to increase *locklist* if *maxappls* is increased, or if the applications being run perform infrequent commits.

You should consider rebinding applications (using the REBIND command) after changing this parameter.

Related concepts:
- "Self tuning memory" in *Performance Guide*

Related reference:
- "locks_held_top - Maximum Number of Locks Held monitor element" in *System Monitor Guide and Reference*
- "lock_escals - Number of Lock Escalations monitor element" in *System Monitor Guide and Reference*
- "REBIND " on page 659
- "maxappls - Maximum number of active applications configuration parameter" in *Performance Guide*
- "maxlocks - Maximum percent of lock list before escalation " on page 1512
- "self_tuning_mem- Self tuning memory configuration parameter" in *Performance Guide*
- "GET DATABASE CONFIGURATION " on page 502
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION " on page 627

locktimeout - Lock timeout

Configuration Type	Database
Parameter Type	Configurable
Default [Range]	-1 [-1; 0 – 32 767]

Unit of Measure	Seconds

This parameter specifies the number of seconds that an application will wait to obtain a lock. This helps avoid global deadlocks for applications.

If you set this parameter to `0`, locks are not waited for. In this situation, if no lock is available at the time of the request, the application immediately receives a -911.

If you set this parameter to `-1`, lock timeout detection is turned off. In this situation a lock will be waited for (if one is not available at the time of the request) until either of the following:

- The lock is granted
- A deadlock occurs.

Recommendation: In a transaction processing (OLTP) environment, you can use an initial starting value of 30 seconds. In a query-only environment you could start with a higher value. In both cases, you should use benchmarking techniques to tune this parameter.

When working with Data Links Manager, if you see lock timeouts in the administration notification log of the Data Links Manager (dlfm) instance, then you should increase the value of *locktimeout*. You should also consider increasing the value of *locklist*.

The value should be set to quickly detect waits that are occurring because of an abnormal situation, such as a transaction that is stalled (possibly as a result of a user leaving their workstation). You should set it high enough so valid lock requests do not time-out because of peak workloads, during which time, there is more waiting for locks.

You can use the database system monitor to help you track the number of times an application (connection) experienced a lock timeout or that a database detected a timeout situation for all applications that were connected.

High values of the *lock_timeout* (number of lock timeouts) monitor element can be caused by:

- Too low a value for this configuration parameter.
- An application (transaction) that is holding locks for an extended period. You can use the database system monitor to further investigate these applications.
- A concurrency problem, that could be caused by lock escalations (from row-level to a table-level lock).

Related concepts:

- "Lock waits and timeouts" in *Performance Guide*

Related reference:

- "lock_timeouts - Number of Lock Timeouts monitor element" in *System Monitor Guide and Reference*
- "GET DATABASE CONFIGURATION " on page 502
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION " on page 627
- "locklist - Maximum storage for lock list " on page 1508
- "maxlocks - Maximum percent of lock list before escalation " on page 1512

maxlocks - Maximum percent of lock list before escalation

Configuration Type	Database	
Parameter Type	Configurable online	
Propagation Class	Immediate	
Default [Range]		
	UNIX	Automatic [1 – 100]
	Windows	Automatic [1 – 100]
Unit of Measure	Percentage	

Lock escalation is the process of replacing row locks with table locks, reducing the number of locks in the list. This parameter defines a percentage of the lock list held by an application that must be filled before the database manager performs escalation. When the number of locks held by any one application reaches this percentage of the total lock list size, lock escalation will occur for the locks held by that application. Lock escalation also occurs if the lock list runs out of space.

The database manager determines which locks to escalate by looking through the lock list for the application and finding the table with the most row locks. If after replacing these with a single table lock, the *maxlocks* value is no longer exceeded, lock escalation will stop. If not, it will continue until the percentage of the lock list held is below the value of *maxlocks*. The *maxlocks* parameter multiplied by the *maxappls* parameter cannot be less than 100.

When this parameter is set to AUTOMATIC, it is enabled for self tuning. This allows the memory tuner to dynamically size the memory area controlled by this parameter as the workload requirements change. Because the memory tuner trades memory resources between different memory consumers, there must be at least two memory consumers enabled for self tuning in order for self tuning to be active.

The value of *locklist* is tuned together with the *maxlocks* parameter, therefore disabling self tuning of the *locklist* parameter automatically disables self tuning of the *maxlocks* parameter. Enabling self tuning of the *locklist* parameter automatically enables self tuning of the *maxlocks* parameter.

Recommendation: The following formula allows you to set *maxlocks* to allow an application to hold twice the average number of locks:

```
   maxlocks = 2 * 100 / maxappls
```

Where `2` is used to achieve twice the average and `100` represents the largest percentage value allowed. If you have only a few applications that run concurrently, you could use the following formula as an alternative to the first formula:

```
   maxlocks = 2 * 100 / (average number of applications running
concurrently)
```

One of the considerations when setting *maxlocks* is to use it in conjunction with the size of the lock list (*locklist*). The actual limit of the number of locks held by an application before lock escalation occurs is:

`maxlocks * locklist * 4 096 / (100 * 48)` on a 32-bit system

```
   maxlocks * locklist * 4 096 / (100 * 80)
```
on a 64-bit system HP-UX/PA-RISC environment

```
   maxlocks * locklist * 4 096 / (100 * 64)
```
on other 64-bit systems

Where `4 096` is the number of bytes in a page, `100` is the largest percentage value allowed for *maxlocks*, and `48` is the number of bytes per lock on a 32-bit system, `80` is the number of bytes per lock on a HP-UX/PA-RISC 64-bit system, and `64` is the number of bytes per lock on other 64-bit systems. If you know that one of your applications requires 1 000 locks, and you do not want lock escalation to occur, then you should choose values for *maxlocks* and *locklist* in this formula so that the result is greater than 1 000. (Using 10 for *maxlocks* and 100 for *locklist*, this formula results in greater than the 1 000 locks needed.)

If *maxlocks* is set too low, lock escalation happens when there is still enough lock space for other concurrent applications. If *maxlocks* is set too high, a few applications can consume most of the lock space, and other applications will have to perform lock escalation. The need for lock escalation in this case results in poor concurrency.

You can use the database system monitor to help you track and tune this configuration parameter.

Related concepts:
- "Self tuning memory" in *Performance Guide*

Related reference:
- "GET DATABASE CONFIGURATION " on page 502
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION " on page 627
- "locklist - Maximum storage for lock list " on page 1508
- "maxappls - Maximum number of active applications configuration parameter" in *Performance Guide*
- "self_tuning_mem- Self tuning memory configuration parameter" in *Performance Guide*

Communications in a partitioned database environment

The following parameters provide information about communications in the partitioned database environment.

conn_elapse - Connection elapse time

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [0–100]
Unit of Measure	Seconds

This parameter specifies the number of seconds within which a TCP/IP connection is to be established between two database partition servers. If the attempt completes within the time specified by this parameter, communications are established. If it fails, another attempt is made to establish communications. If the connection is attempted the number of times specified by the *max_connretries* parameter and always times out, an error is issued.

Related reference:

- "max_connretries - Node connection retries " on page 1516
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

fcm_num_buffers - Number of FCM buffers

Configuration Type	Database manager
Applies To	• Database server with local and remote clients • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	**32-bit platforms** Automatic [Automatic, 128 — *65 300*] **64-bit platforms** Automatic [Automatic, 128 — *524 288*] • Database server with local and remote clients: the default is 1 024 • Database server with local clients: the default is 512 • Partitioned database server with local and remote clients: the default is 4 096 On single-partition database systems, this parameter is not used if the *intra_parallel* parameter is not active.

This parameter specifies the number of 4 KB buffers that are used for internal communications (messages) both among and within database servers.

If you have multiple logical nodes on the same machine, you might find it necessary to increase the value of this parameter. You might also find it necessary to increase the value of this parameter if you run out of message buffers because of the number of users on the system, the number of database partition servers on the system, or the complexity of the applications.

If you are using multiple logical nodes, one pool of *fcm_num_buffers* buffers is shared by all the multiple logical nodes on the same machine. The size of the buffer pool will be determined by multiplying the fcm_num_buffersvalue times the number of logical nodes on that physical machine. Re-examine the value you are using; consider how many FCM buffers in total will be allocated on the machine (or machines) where the multiple logical nodes reside.

Related concepts:

- "Fast communications manager (FCM) communications" on page 289

Related reference:

- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

fcm_num_channels - Number of FCM channels

Configuration Type
Database manager

Applies To

- Database server with local and remote clients
- Database server with local clients
- Partitioned database server with local and remote clients
- Satellite database server with local clients

Parameter Type
Configurable Online

Propagation Class
Immediate

Default [Range]

UNIX 32-bit platforms
Automatic, with starting values of 256, 512, 2 048 [128 — 120 000]

UNIX 64-bit platforms
Automatic, with starting values of 256, 512, 2 048 [128 — 524 288]

Windows 32-bit
Automatic, with a starting value 10 000 [128 — 120 000]

Windows 64-bit
Automatic, with starting values of 256, 512, 2 048 [128 — 524 288]

- For database server with local and remote clients, the starting value is 512.
- For database server with local clients, the starting value is 256.
- For partitioned database server with local and remote clients, the starting value is 2 048.

On non-partitioned database systems, the *intra_parallel* parameter must be active before *fcm_num_channels* can be used.

An FCM channel represents a logical communication end point between EDUs running in the DB2 engine. Both control flows (request and reply) and data flows (table queue data) rely on channels to transfer data between partitions. This parameter specifies the number of FCM channels for each database partition.

Related reference:

- "ch_free - Channels Currently Free monitor element" in *System Monitor Guide and Reference*
- "ch_free_bottom - Minimum Channels Free monitor element" in *System Monitor Guide and Reference*

max_connretries - Node connection retries

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	5 [0–100]

If the attempt to establish communication between two database partition servers fails (for example, the value specified by the *conn_elapse* parameter is reached), *max_connretries* specifies the number of connection retries that can be made to a database partition server. If the value specified for this parameter is exceeded, an error is returned.

Related reference:

- "conn_elapse - Connection elapse time " on page 1513
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

max_time_diff - Maximum time difference among nodes

Configuration Type	Database manager
Applies To	Partitioned database server with local and remote clients
Parameter Type	Configurable
Default [Range]	60 [1–1 440]
Unit of Measure	Minutes

Each database partition server has its own system clock. This parameter specifies the maximum time difference, in minutes, that is permitted among the database partition servers listed in the node configuration file.

If two or more database partition servers are associated with a transaction, and their clocks are not synchronized to within the time specified by this parameter, the transaction is rejected and an SQLCODE is returned. (The transaction is rejected only if data modification is associated with it.)

DB2 uses *Coordinated Universal Time* (UTC), so different time zones are not a consideration when you set this parameter. The Coordinated Universal Time is the same as Greenwich Mean Time.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

start_stop_time - Start and stop timeout

Configuration Type	Database manager
Applies To	Database server with local and remote clients
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	10 [1 — 1 440]
Unit of Measure	Minutes

This parameter specifies the time, in minutes, within which all database partition servers must respond to a DB2START or a DB2STOP command. It is also used as the timeout value during an ADD DBPARTITIONNUM operation.

Database partition servers that do not respond to a DB2START command within the specified time send a message to the `db2start` error log in the `log` subdirectory of the `sqllib` subdirectory of the home directory for the instance. You should issue a DB2STOP on these nodes before restarting them.

Database partition servers that do not respond to a DB2STOP command within the specified time send a message to the `db2stop` error log in the `log` subdirectory of the `sqllib` subdirectory of the home directory for the instance. You can either issue `db2stop` for each database partition server that does not respond, or for all of them. (Those that are already stopped will return stating that they are stopped.)

If a `db2start` or `db2stop` operation in a multi-partition database is not completed within the value specified by the *start_stop_timeout* database manager configuration parameter, the database partitions that have timed out will be killed internally. Environments with many database partitions with a low value for *start_stop_timeout* might experience this behavior. To resolve this behavior, increase the value of *start_stop_timeout*.

When adding a new database partition using one of the DB2START, START DATABASE MANAGER, or ADD DBPARTITIONNUM commands, the add database partition operation must determine whether or not each database in the instance is enabled for automatic storage. This is done by communicating with the catalog partition for each database. If automatic storage is enabled, the storage path definitions are retrieved as part of that communication. Likewise, if system temporary table spaces are to be created with the database partitions, the operation might have to communicate with another database partition server to retrieve the table space definitions for the database partitions that reside on that server. These factors should be considered when determining the value of the *start_stop_time* parameter.

Related reference:
- "ADD DBPARTITIONNUM " on page 435
- "GET DATABASE MANAGER CONFIGURATION " on page 508
- "RESET DATABASE MANAGER CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE MANAGER CONFIGURATION " on page 629

autorestart - Auto restart enable

Configuration Type	Database
Parameter Type	Configurable Online
Propagation Class	Immediate
Default [Range]	On [On; Off]

When this parameter is set on, the database manager automatically calls the restart database utility, if needed, when an application connects to a database. *Crash recovery* is the operation performed by the restart database utility. It is performed if the database terminated abnormally while applications were connected to it. An abnormal termination of the database could be caused by a power failure or a system software failure. It applies any committed transactions that were in the database buffer pool but were not written to disk at the time of the failure. It also backs out any uncommitted transactions that might have been written to disk.

If *autorestart* is not enabled, then an application that attempts to connect to a database which needs to have crash recovery performed (needs to be restarted) will receive a SQL1015N error. In this case, the application can call the restart database utility, or you can restart the database by selecting the restart operation of the recovery tool.

Related concepts:
- "Crash recovery" on page 1533

Related reference:
- "RESTART DATABASE " on page 589
- "GET DATABASE CONFIGURATION " on page 502
- "RESET DATABASE CONFIGURATION command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION " on page 627

database_consistent - Database is consistent

Configuration Type	Database
Parameter Type	Informational

This parameter indicates whether the database is in a consistent state.

YES indicates that all transactions have been committed or rolled back so that the data is consistent. If the system "crashes" while the database is consistent, you do not need to take any special action to make the database usable.

NO indicates that a transaction is pending or some other task is pending on the database and the data is not consistent at this point. If the system "crashes" while the database is not consistent, you will need to restart the database using the RESTART DATABASE command to make the database usable.

Related reference:
- "GET DATABASE CONFIGURATION " on page 502

nodetype - Machine node type

Configuration Type	Database manager
Applies to	• Database server with local and remote clients • Client • Database server with local clients • Partitioned database server with local and remote clients
Parameter Type	Informational

This parameter provides information about the DB2 products which you have installed on your machine and, as a result, information about the type of database manager configuration. The following are the possible values returned by this parameter and the products associated with that node type:
- **Database server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers.
- **Client** – a database client capable of accessing remote database servers.
- **Database server with local clients** – a DB2 relational database management system, supporting local database clients and capable of accessing other, remote database servers.
- **Partitioned database server with local and remote clients** – a DB2 server product, supporting local and remote database clients, and capable of accessing other remote database servers, and capable of parallelism.

Related reference:
- "GET DATABASE MANAGER CONFIGURATION " on page 508

restrict_access - Database has restricted access configuration parameter

Configuration Type	Database
Parameter Type	Informational

This parameter indicates whether the database was created using the restrictive set of default actions. In other words if it was created with the RESTRICTIVE clause in the CREATE DATABASE command.

YES The RESTRICTIVE clause was used in the CREATE DATABASE command when this database was created.

NO The RESTRICTIVE clause was not used in the CREATE DATABASE command when this database was created.

Part 16. Special registers

Chapter 49. Security-related special registers

Special registers

A *special register* is a storage area that is defined for an application process by the database manager. It is used to store information that can be referenced in SQL statements. A reference to a special register is a reference to a value provided by the current server. If the value is a string, its CCSID is a default CCSID of the current server. The special registers can be referenced as follows:

```
►►──┬─┬─CURRENT CLIENT_ACCTNG─────┬──────────────────────┬──────────────────────────►◄
    │ └─CLIENT ACCTNG─────────────┘                      │
    ├─┬─CURRENT CLIENT_APPLNAME─┬────────────────────────┤
    │ └─CLIENT APPLNAME─────────┘                        │
    ├─┬─CURRENT CLIENT_USERID─┬──────────────────────────┤
    │ └─CLIENT USERID─────────┘                          │
    ├─┬─CURRENT CLIENT_WRKSTNNAME─┬──────────────────────┤
    │ └─CLIENT WRKSTNNAME─────────┘                      │
    ├─┬─CURRENT DATE──────┬──────────────────────────────┤
    │ │             (1)   │                              │
    │ └─CURRENT_DATE──────┘                              │
    ├─CURRENT DBPARTITIONNUM─────────────────────────────┤
    ├─CURRENT DEFAULT TRANSFORM GROUP────────────────────┤
    ├─CURRENT DEGREE─────────────────────────────────────┤
    ├─CURRENT EXPLAIN MODE───────────────────────────────┤
    ├─CURRENT EXPLAIN SNAPSHOT───────────────────────────┤
    ├─CURRENT FEDERATED ASYNCHRONY───────────────────────┤
    ├─CURRENT IMPLICIT XMLPARSE OPTION───────────────────┤
    ├─CURRENT ISOLATION──────────────────────────────────┤
    ├─CURRENT LOCK TIMEOUT───────────────────────────────┤
    ├─CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION────┤
    ├─CURRENT PACKAGE PATH───────────────────────────────┤
    ├─┬─CURRENT PATH──────┬──────────────────────────────┤
    │ │             (1)   │                              │
    │ └─CURRENT_PATH──────┘                              │
    ├─CURRENT QUERY OPTIMIZATION─────────────────────────┤
    ├─CURRENT REFRESH AGE────────────────────────────────┤
    ├─┬─CURRENT SCHEMA──────┬────────────────────────────┤
    │ │               (1)   │                            │
    │ └─CURRENT_SCHEMA──────┘                            │
    ├─┬─CURRENT SERVER──────┬────────────────────────────┤
    │ │               (1)   │                            │
    │ └─CURRENT_SERVER──────┘                            │
    ├─┬─CURRENT TIME──────┬──────────────────────────────┤
    │ │             (1)   │                              │
    │ └─CURRENT_TIME──────┘                              │
    ├─┬─CURRENT TIMESTAMP──────┬─────────────────────────┤
    │ │                  (1)   │                         │
    │ └─CURRENT_TIMESTAMP──────┘                         │
    ├─┬─CURRENT TIMEZONE──────┬──────────────────────────┤
    │ │                 (1)   │                          │
    │ └─CURRENT_TIMEZONE──────┘                          │
    ├─┬─CURRENT USER──────┬──────────────────────────────┤
    │ │             (1)   │                              │
    │ └─CURRENT_USER──────┘                              │
    ├─┬─SESSION_USER─┬───────────────────────────────────┤
    │ └─USER─────────┘                                   │
    └─SYSTEM_USER────────────────────────────────────────┘
```

Notes:

1 The SQL 1999 Core standard uses the form with the underscore.

Some special registers can be updated using the SET statement. The following table shows which of the special registers can be updated.

Table 170. Special Registers

Special Register	Updatable
CURRENT CLIENT_ACCTNG	No

Table 170. Special Registers (continued)

Special Register	Updatable
CURRENT CLIENT_APPLNAME	No
CURRENT CLIENT_USERID	No
CURRENT CLIENT_WRKSTNNAME	No
CURRENT DATE	No
CURRENT DBPARTITIONNUM	No
CURRENT DEFAULT TRANSFORM GROUP	Yes
CURRENT DEGREE	Yes
CURRENT EXPLAIN MODE	Yes
CURRENT EXPLAIN SNAPSHOT	Yes
CURRENT FEDERATED ASYNCHRONY	Yes
CURRENT IMPLICIT XMLPARSE OPTION	Yes
CURRENT ISOLATION	Yes
CURRENT LOCK TIMEOUT	Yes
CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION	Yes
CURRENT PACKAGE PATH	Yes
CURRENT PATH	Yes
CURRENT QUERY OPTIMIZATION	Yes
CURRENT REFRESH AGE	Yes
CURRENT SCHEMA	Yes
CURRENT SERVER	No
CURRENT TIME	No
CURRENT TIMESTAMP	No
CURRENT TIMEZONE	No
CURRENT USER	No
SESSION_USER	Yes
SYSTEM_USER	No
USER	Yes

When a special register is referenced in a routine, the value of the special register in the routine depends on whether the special register is updatable or not. For non-updatable special registers, the value is set to the default value for the special register. For updatable special registers, the initial value is inherited from the invoker of the routine and can be changed with a subsequent SET statement inside the routine.

CURRENT CLIENT_APPLNAME

The CURRENT CLIENT_APPLNAME (or CLIENT APPLNAME) special register contains the value of the application name from the client information specified for this connection. The data type of the register is VARCHAR(255). The default value of this register is an empty string.

The value of the application name can be changed by using the Set Client Information (sqleseti) API.

Note that the value provided via the sqleseti API is in the application code page, and the special register value is stored in the database code page. Depending on the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

Example: Select which departments are allowed to use the application being used in this connection.

```
SELECT DEPT
  FROM DEPT_APPL_MAP
  WHERE APPL_NAME = CURRENT CLIENT_APPLNAME
```

CURRENT CLIENT_USERID

The CURRENT CLIENT_USERID (or CLIENT USERID) special register contains the value of the client user ID from the client information specified for this connection. The data type of the register is VARCHAR(255). The default value of this register is an empty string.

The value of the client user ID can be changed by using the Set Client Information (sqleseti) API.

Note that the value provided via the sqleseti API is in the application code page, and the special register value is stored in the database code page. Depending on the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

Example: Find out in which department the current client user ID works.

```
SELECT DEPT
  FROM DEPT_USERID_MAP
  WHERE USER_ID = CURRENT CLIENT_USERID
```

CURRENT CLIENT_WRKSTNNAME

The CURRENT CLIENT_WRKSTNNAME (or CLIENT WRKSTNNAME) special register contains the value of the workstation name from the client information specified for this connection. The data type of the register is VARCHAR(255). The default value of this register is an empty string.

The value of the workstation name can be changed by using the Set Client Information (sqleseti) API.

Note that the value provided via the sqleseti API is in the application code page, and the special register value is stored in the database code page. Depending on the data values used when setting the client information, truncation of the data value stored in the special register may occur during code page conversion.

Example: Get the workstation name being used for this connection.

```
VALUES (CURRENT CLIENT_WRKSTNNAME)
  INTO :WS_NAME
```

CURRENT SERVER

The CURRENT SERVER (or CURRENT_SERVER) special register specifies a VARCHAR(18) value that identifies the current application server. The register contains the actual name of the application server, not an alias.

CURRENT SERVER can be changed through the CONNECT statement, but only under certain conditions.

When used in an SQL statement inside a routine, CURRENT SERVER is not inherited from the invoking statement.

Example: Set the host variable APPL_SERVE (VARCHAR(18)) to the name of the application server to which the application is connected.

```
VALUES CURRENT SERVER INTO :APPL_SERVE
```

Related reference:

- “CONNECT (Type 1) ” on page 1159

CURRENT DBPARTITIONNUM

The CURRENT DBPARTITIONNUM special register specifies an INTEGER value that identifies the coordinator node number for the statement. For statements issued from an application, the coordinator is the database partition to which the application connects. For statements issued from a routine, the coordinator is the database partition from which the routine is invoked.

When used in an SQL statement inside a routine, CURRENT DBPARTITIONNUM is never inherited from the invoking statement.

CURRENT DBPARTITIONNUM returns 0 if the database instance is not defined to support database partitioning. (In other words, if there is no db2nodes.cfg file. For partitioned databases, the db2nodes.cfg file exists and contains database partition definitions.)

CURRENT DBPARTITIONNUM can be changed through the CONNECT statement, but only under certain conditions.

For compatibility with versions earlier than Version 8, the keyword NODE can be substituted for DBPARTITIONNUM.

Example: Set the host variable APPL_NODE (integer) to the number of the database partition to which the application is connected.

```
VALUES CURRENT DBPARTITIONNUM
  INTO :APPL_NODE
```

Related reference:

- “CONNECT (Type 1) ” on page 1159

CURRENT LOCK TIMEOUT

The CURRENT LOCK TIMEOUT special register specifies the number of seconds to wait for a lock before returning an error indicating that a lock cannot be obtained. This special register impacts row, table, index key, MDC block, and XML path (XPath) locks. The data type of the register is INTEGER.

Valid values for the CURRENT LOCK TIMEOUT special register are integers between -1 and 32767, inclusive. This special register can also be set to the null value. A value of -1 specifies that timeouts are not to take place, and that the application is to wait until the lock is released or a deadlock is detected. A value of 0 specifies that the application is not to wait for a lock; if a lock cannot be obtained, an error is to be returned immediately.

The value of the CURRENT LOCK TIMEOUT special register can be changed by invoking the SET CURRENT LOCK TIMEOUT statement. Its initial value is null; in this case, the current value of the *locktimeout* database configuration parameter is used when waiting for a lock, and this value will be returned for the special register.

Related reference:

- "SET CURRENT LOCK TIMEOUT statement" in *SQL Reference, Volume 2*

CURRENT SCHEMA

The CURRENT SCHEMA (or CURRENT_SCHEMA) special register specifies a VARCHAR(128) value that identifies the schema name used to qualify database object references, where applicable, in dynamically prepared SQL statements. For compatibility with DB2 for OS/390, CURRENT SQLID (or CURRENT_SQLID) can be specified in place of CURRENT SCHEMA.

The initial value of CURRENT SCHEMA is the authorization ID of the current session user. The value can be changed by invoking the SET SCHEMA statement.

The QUALIFIER bind option controls the schema name used to qualify database object references, where applicable, for static SQL statements.

Example: Set the schema for object qualification to 'D123'.

```
SET CURRENT SCHEMA =  'D123'
```

SESSION_USER

The SESSION_USER special register specifies the authorization ID that is to be used for the current session. The value of this register is used for authorization checking of dynamic SQL statements when DYNAMICRULES run behavior is in effect for the package. The data type of the register is VARCHAR(128).

The initial value of SESSION_USER for a new connection is the same as the value of the SYSTEM_USER special register. Its value can be changed by invoking the SET SESSION AUTHORIZATION statement.

SESSION_USER is a synonym for the USER special register.

Example: Determine what routines can be executed using dynamic SQL. Assume DYNAMICRULES run behavior is in effect for the package that will issue the dynamic SQL statement that invokes the routine.

```
SELECT SCHEMA, SPECIFICNAME FROM SYSCAT.ROUTINEAUTH
  WHERE GRANTEE = SESSION_USER
  AND EXECUTEAUTH IN ('Y', 'G')
```

Related reference:

- "SET SESSION AUTHORIZATION statement" in *SQL Reference, Volume 2*

SYSTEM_USER

The SYSTEM_USER special register specifies the authorization ID of the user that connected to the database. The value of this register can only be changed by connecting as a user with a different authorization ID. The data type of the register is VARCHAR(128).

See "Example" in the description of the SET SESSION AUTHORIZATION statement.

Related reference:

- "SET SESSION AUTHORIZATION statement" in *SQL Reference, Volume 2*

USER

The USER special register specifies the run-time authorization ID passed to the database manager when an application starts on a database. The data type of the register is VARCHAR(128).

When used in an SQL statement inside a routine, USER is not inherited from the invoking statement.

Example: Select all notes from the `IN_TRAY` table that were placed there by the user.

```
SELECT * FROM IN_TRAY
  WHERE SOURCE = USER
```

Part 17. Recovery considerations

Chapter 50. Crash Recovery and Database Logs

Crash recovery

Transactions (or units of work) against a database can be interrupted unexpectedly. If a failure occurs before all of the changes that are part of the unit of work are completed and committed, the database is left in an inconsistent and unusable state. *Crash recovery* is the process by which the database is moved back to a consistent and usable state. This is done by rolling back incomplete transactions and completing committed transactions that were still in memory when the crash occurred (Figure 39). When a database is in a consistent and usable state, it has attained what is known as a ″point of consistency″.

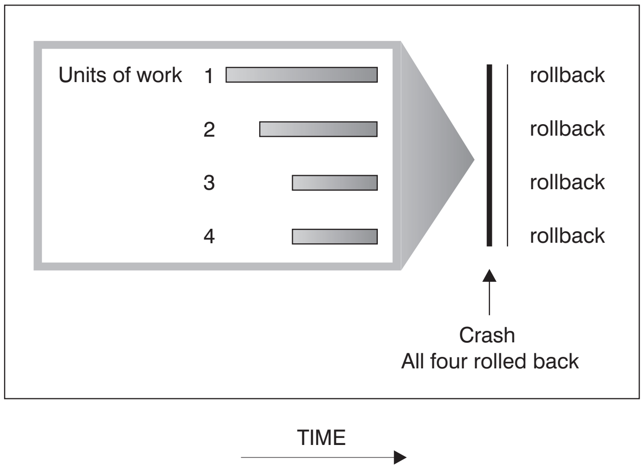


Figure 39. Rolling Back Units of Work (Crash Recovery)

A *transaction failure* results from a severe error or condition that causes the database or the database manager to end abnormally. Partially completed units of work, or UOW that have not been flushed to disk at the time of failure, leave the database in an inconsistent state. Following a transaction failure, the database must be recovered. Conditions that can result in transaction failure include:

- A power failure on the machine, causing the database manager and the database partitions on it to go down
- A hardware failure such as memory corruption, or disk, CPU, or network failure.
- A serious operating system error that causes DB2 to go down
- An application terminating abnormally.

If you want the rollback of incomplete units of work to be done automatically by the database manager, enable the automatic restart (*autorestart*) database configuration parameter by setting it to `ON`. (This is the default value.) If you do not want automatic restart behavior, set the *autorestart* database configuration parameter to `OFF`. As a result, you will need to issue the RESTART DATABASE command when a database failure occurs. If the database I/O was suspended before the crash occurred, you must specify the WRITE RESUME option of the

RESTART DATABASE command in order for the crash recovery to continue. The administration notification log records when the database restart operation begins.

If crash recovery is applied to a database that is enabled for forward recovery (that is, the *logarchmeth1* configuration parameter is not set to `OFF`), and an error occurs during crash recovery that is attributable to an individual table space, that table space will be taken offline, and cannot be accessed until it is repaired. Crash recovery continues. At the completion of crash recovery, the other table spaces in the database will be accessible, and connections to the database can be established. However, if the table space that is taken offline is the table space that contains the system catalogs, it must be repaired before any connections will be permitted.

Related reference:

- "autorestart - Auto restart enable " on page 1518
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*

Understanding recovery logs

All databases have logs associated with them. These logs keep records of database changes. If a database needs to be restored to a point beyond the last full, offline backup, logs are required to roll the data forward to the point of failure.

There are two types of DB2 logging: *circular* and *archive*. Each provides a different level of recovery capability:

- *Circular* logging is the default behavior when a new database is created. (The *logarchmeth1* and *logarchmeth2* database configuration parameters are set to `OFF`.) With this type of logging, only full, offline backups of the database are allowed. The database must be offline (inaccessible to users) when a full backup is taken. As the name suggests, circular logging uses a "ring" of online logs to provide recovery from transaction failures and system crashes. The logs are used and retained only to the point of ensuring the integrity of current transactions. Circular logging does not allow you to roll a database forward through transactions performed after the last full backup operation. All changes occurring since the last backup operation are lost. Since this type of restore operation recovers your data to the specific point in time at which a full backup was taken, it is called *version recovery*.

  Figure 40 on page 1535 shows that the active log uses a ring of log files when circular logging is active.

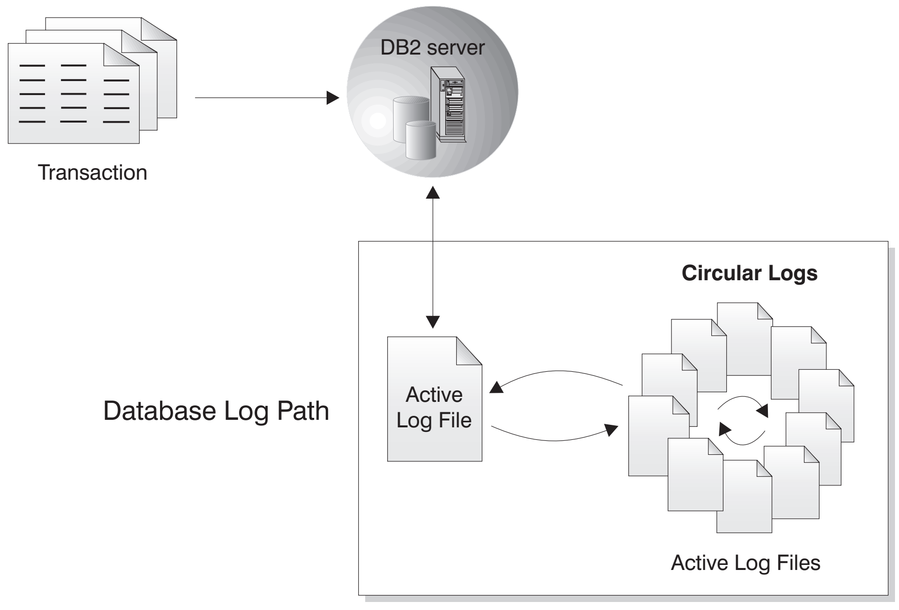


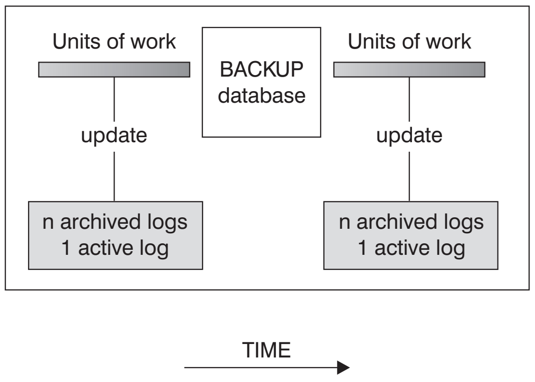
Figure 40. Circular Logging

Active logs are used during crash recovery to prevent a failure (system power or application error) from leaving a database in an inconsistent state. Active logs are located in the database log path directory.

- *Archive* logging is used specifically for rollforward recovery. Archived logs are logs that were active but are no longer required for crash recovery. Use the *logarchmeth1* database configuration parameter to enable archive logging.

The advantage of choosing archive logging is that rollforward recovery can use both archived logs and active logs to restore a database either to the end of the logs, or to a specific point in time. The archived log files can be used to recover changes made after the backup was taken. This is different from circular logging where you can only recover to the time of the backup, and all changes made after that are lost.

Taking online backups is only supported if the database is configured for archive logging. During an online backup operation, all activities against the database are logged. When an online backup image is restored, the logs must be rolled forward at least to the point in time at which the backup operation completed. For this to happen, the logs must have been archived and made available when the database is restored. After an online backup is complete, DB2 forces the currently active log to be closed, and as a result, it will be archived. This ensures that your online backup has a complete set of archived logs available for recovery.



Logs are used between backups to track the changes to the databases.

Figure 41. Active and Archived Database Logs in Rollforward Recovery. There can be more than one active log in the case of a long-running transaction.

The following database configuration parameters allow you to change where archived logs are stored: The *newlogpath* parameter, and the *logarchmeth1* and *logarchmeth2* parameters. Changing the *newlogpath* parameter also affects where active logs are stored.

To determine which log *extents* in the database log path directory are archived logs, check the value of the *loghead* database configuration parameter. This parameter indicates the lowest numbered log that is active. Those logs with sequence numbers less than *loghead* are archived logs and can be moved. You can check the value of this parameter by using the Control Center; or, by using the command line processor and the GET DATABASE CONFIGURATION command to view the ″First active log file″. For more information about this configuration parameter, see the *Performance Guide* book.

Related concepts:

- "Log mirroring" in *Data Recovery and High Availability Guide and Reference*

Related reference:

- "Configuration parameters for database logging" in *Data Recovery and High Availability Guide and Reference*
- "logarchmeth1 - Primary log archive method configuration parameter" in *Performance Guide*
- "loghead - First active log file configuration parameter" in *Performance Guide*
- "User exit for database recovery" in *Data Recovery and High Availability Guide and Reference*

Chapter 51. Application processes, concurrency, and recovery

All SQL programs execute as part of an *application process* or agent. An application process involves the execution of one or more programs, and is the unit to which the database manager allocates resources and locks. Different application processes may involve the execution of different programs, or different executions of the same program.

More than one application process may request access to the same data at the same time. *Locking* is the mechanism used to maintain data integrity under such conditions, preventing, for example, two application processes from updating the same row of data simultaneously.

The database manager acquires locks to prevent uncommitted changes made by one application process from being accidentally perceived by any other process. The database manager releases all locks it has acquired and retained on behalf of an application process when that process ends. However, an application process can explicitly request that locks be released sooner. This is done using a *commit* operation, which releases locks acquired during the unit of work and also commits database changes made during the unit of work.

The database manager provides a means of backing out uncommitted changes made by an application process. This might be necessary in the event of a failure on the part of an application process, or in the case of a deadlock, or a lock time-out situation. An application process can explicitly request that its database changes be backed out. This is done using a *rollback* operation.

A *unit of work* is a recoverable sequence of operations within an application process. A unit of work is initiated when an application process is started, or when the previous unit of work is ended by something other than the termination of the application process. A unit of work is ended by a commit operation, a rollback operation, or the end of an application process. A commit or rollback operation affects only the database changes made within the unit of work it is ending.

As long as these changes remain uncommitted, other application processes are unable to perceive them, and they can be backed out. This is not true, however, when the isolation level is uncommitted read (UR). Once committed, these database changes are accessible by other application processes and can no longer be backed out through a rollback.

Both DB2 call level interface (CLI) and embedded SQL allow for a connection mode called *concurrent transactions*, which supports multiple connections, each of which is an independent transaction. An application can have multiple concurrent connections to the same database.

Locks acquired by the database manager on behalf of an application process are held until the end of a unit of work. This is not true, however, when the isolation level is cursor stability (CS, in which the lock is released as the cursor moves from row to row) or uncommitted read (UR, in which locks are not obtained).

An application process is never prevented from performing operations because of its own locks. However, if an application uses concurrent transactions, the locks from one transaction may affect the operation of a concurrent transaction.

The initiation and the termination of a unit of work define points of consistency within an application process. For example, a banking transaction may involve the transfer of funds from one account to another. Such a transaction would require that these funds be subtracted from the first account, and then added to the second account. Following the subtraction step, the data is inconsistent. Only after the funds have been added to the second account is consistency reestablished. When both steps are complete, the commit operation can be used to end the unit of work, thereby making the changes available to other application processes. If a failure occurs before the unit of work ends, the database manager will roll back uncommitted changes to restore the data consistency that it assumes existed when the unit of work was initiated.

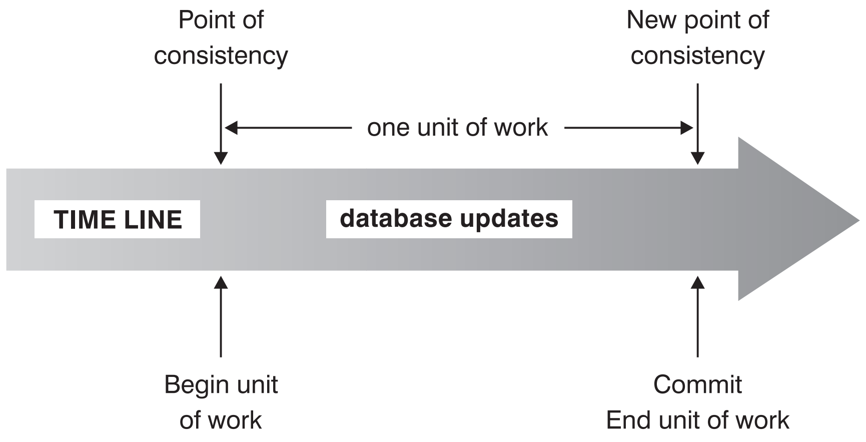


Figure 42. Unit of Work with a COMMIT Statement

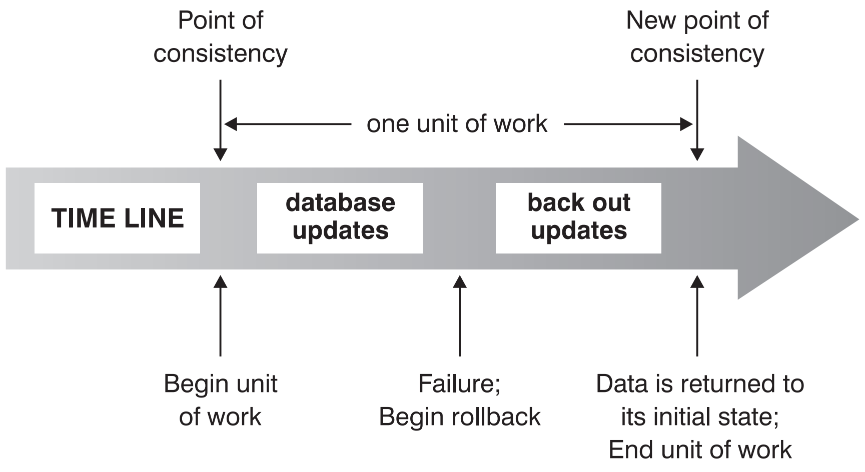


Figure 43. Unit of Work with a ROLLBACK Statement

Related concepts:

- "Isolation levels" in *SQL Reference, Volume 1*

Chapter 52. Recovering from transaction failures in a partitioned database environment

If a transaction failure occurs in a partitioned database environment, database recovery is usually necessary on both the failed database partition server and any other database partition server that was participating in the transaction:

- Crash recovery occurs on the failed database partition server after the antecedent condition is corrected.
- *Database partition failure recovery* on the other (still active) database partition servers occurs immediately after the failure has been detected.

In a partitioned database environment, the database partition server on which an application is submitted is the coordinator partition, and the first agent that works for the application is the coordinator agent. The coordinator agent is responsible for distributing work to other database partition servers, and it keeps track of which ones are involved in the transaction. When the application issues a COMMIT statement for a transaction, the coordinator agent commits the transaction by using the two-phase commit protocol. During the first phase, the coordinator partition distributes a PREPARE request to all the other database partition servers that are participating in the transaction. These servers then respond with one of the following:

READ-ONLY	No data change occurred at this server
YES	Data change occurred at this server
NO	Because of an error, the server is not prepared to commit

If one of the servers responds with a `NO`, the transaction is rolled back. Otherwise, the coordinator partition begins the second phase.

During the second phase, the coordinator partition writes a COMMIT log record, then distributes a COMMIT request to all the servers that responded with a `YES`. After all the other database partition servers have committed, they send an acknowledgment of the COMMIT to the coordinator partition. The transaction is complete when the coordinator agent has received all COMMIT acknowledgments from all the participating servers. At this point, the coordinator agent writes a FORGET log record.

Transaction failure recovery on an active database partition server

If any database partition server detects that another server is down, all work that is associated with the failed database partition server is stopped:

- If the still active database partition server is the coordinator partition for an application, and the application was running on the failed database partition server (and not ready to COMMIT), the coordinator agent is interrupted to do failure recovery. If the coordinator agent is in the second phase of COMMIT processing, SQL0279N is returned to the application, which in turn loses its database connection. Otherwise, the coordinator agent distributes a ROLLBACK request to all other servers participating in the transaction, and SQL1229N is returned to the application.

- If the failed database partition server was the coordinator partition for the application, agents that are still working for the application on the active servers are interrupted to do failure recovery. The current transaction is rolled back locally on each server, unless it has been prepared and is waiting for the transaction outcome. In this situation, the transaction is left in doubt on the active database partition servers, and the coordinator partition is not aware of this (because it is not available).
- If the application connected to the failed database partition server (before it failed), but neither the local database partition server nor the failed database partition server is the coordinator partition, agents working for this application are interrupted. The coordinator partition will either send a ROLLBACK or a disconnect message to the other database partition servers. The transaction will only be indoubt on database partition servers that are still active if the coordinator partition returns SQL0279.

Any process (such as an agent or deadlock detector) that attempts to send a request to the failed server is informed that it cannot send the request.

Transaction failure recovery on the failed database partition server

If the transaction failure causes the database manager to end abnormally, you can issue the `db2start` command with the RESTART option to restart the database manager once the database partition has been restarted. If you cannot restart the database partition, you can issue `db2start` to restart the database manager on a different database partition.

If the database manager ends abnormally, database partitions on the server can be left in an inconsistent state. To make them usable, crash recovery can be triggered on a database partition server:

- Explicitly, through the RESTART DATABASE command
- Implicitly, through a CONNECT request when the *autorestart* database configuration parameter has been set to `ON`

Crash recovery reapplies the log records in the active log files to ensure that the effects of all complete transactions are in the database. After the changes have been reapplied, all uncommitted transactions are rolled back locally, *except* for indoubt transactions. There are two types of indoubt transaction in a partitioned database environment:

- On a database partition server that is not the coordinator partition, a transaction is in doubt if it is prepared but not yet committed.
- On the coordinator partition, a transaction is in doubt if it is committed but not yet logged as complete (that is, the FORGET record is not yet written). This situation occurs when the coordinator agent has not received all the COMMIT acknowledgments from all the servers that worked for the application.

Crash recovery attempts to resolve all the indoubt transactions by doing one of the following. The action that is taken depends on whether the database partition server was the coordinator partition for an application:

- If the server that restarted is not the coordinator partition for the application, it sends a query message to the coordinator agent to discover the outcome of the transaction.
- If the server that restarted *is* the coordinator partition for the application, it sends a message to all the other agents (subordinate agents) that the coordinator agent is still waiting for COMMIT acknowledgments.

It is possible that crash recovery might not be able to resolve all the indoubt transactions (for example, some of the database partition servers might not be available). In this situation, the SQL warning message SQL1061W is returned. Because indoubt transactions hold resources, such as locks and active log space, it is possible to get to a point where no changes can be made to the database because the active log space is being held up by indoubt transactions. For this reason, you should determine whether indoubt transactions remain after crash recovery, and recover all database partition servers that are required to resolve the indoubt transactions as quickly as possible.

If one or more servers that are required to resolve an indoubt transaction cannot be recovered in time, and access is required to database partitions on other servers, you can manually resolve the indoubt transaction by making an heuristic decision. You can use the LIST INDOUBT TRANSACTIONS command to query, commit, and roll back the indoubt transaction on the server.

Note: The LIST INDOUBT TRANSACTIONS command is also used in a distributed transaction environment. To distinguish between the two types of indoubt transactions, the *originator* field in the output that is returned by the LIST INDOUBT TRANSACTIONS command displays one of the following:

- DB2 Enterprise Server Edition, which indicates that the transaction originated in a partitioned database environment.
- XA, which indicates that the transaction originated in a distributed environment.

Identifying the failed database partition server

When a database partition server fails, the application will typically receive one of the following SQLCODEs. The method for detecting which database manager failed depends on the SQLCODE received:

SQL0279N
This SQLCODE is received when a database partition server involved in a transaction is terminated during COMMIT processing.

SQL1224N
This SQLCODE is received when the database partition server that failed is the coordinator partition for the transaction.

SQL1229N
This SQLCODE is received when the database partition server that failed is not the coordinator partition for the transaction.

Determining which database partition server failed is a two-step process. The SQLCA associated with SQLCODE SQL1229N contains the node number of the server that detected the error in the sixth array position of the *sqlerrd* field. (The node number that is written for the server corresponds to the node number in the db2nodes.cfg file.) On the database partition server that detects the error, a message that indicates the node number of the failed server is written to the administration notification log.

Note: If multiple logical nodes are being used on a processor, the failure of one logical node can cause other logical nodes on the same processor to fail.

Related concepts:

- "Error recovery during two-phase commit" in *Administration Guide: Planning*

- “Two-phase commit” in *Administration Guide: Planning*

Related tasks:

- “Resolving indoubt transactions manually” in *Administration Guide: Planning*

Related reference:

- “db2start - Start DB2 command” in *Command Reference*
- “LIST INDOUBT TRANSACTIONS command” in *Command Reference*

Part 18. Appendixes

Notices

Contacting IBM

To contact IBM in your country or region, check the IBM Directory of Worldwide Contacts at http://www.ibm.com/planetwide

To learn more about DB2 products, go to http://www.ibm.com/software/data/db2/.

Index

Printed in USA

SC10-4313-00