DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Data Movement Utilities Guide and Reference

SC10-4227-00

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Data Movement Utilities Guide and Reference

SC10-4227-00

Edition Notice

Contents

About This Book

This book provides information about, and shows you how to use, the following DB2 database for Linux™, UNIX®, and Windows® data movement utilities:

- The import and export utilities move data between a table or view and another database or spreadsheet program; between DB2 databases; and between DB2 databases and host databases using DB2 Connect™. The export utility moves data from a database into operating system files; you can then use those files to import or load that data into another database.
- The load utility moves data into tables, extends existing indexes, and generates statistics. Load moves data much faster than the import utility when large amounts of data are involved. Data unloaded using the export utility can be loaded with the load utility.
- When the load utility is used in a partitioned database environment, large amounts of data can be distributed and loaded into different database partitions.
- DataPropagator is a component of the DB2 database system that allows automatic copying of table updates to other tables in other DB2 relational databases.

Other vendor's products that move data in and out of databases are also available, but are not discussed in this book.

Who Should Use this Book

This manual is for database administrators, application programmers, and other DB2 users who perform the following tasks:

- Load data into DB2 tables from operating system files
- Move data between DB2 databases, and between DB2 and other applications (for example, spreadsheets)
- Archive data

It is assumed that you are familiar with the DB2 database system, Structured Query Language (SQL), and with the operating system environment in which the DB2 database is running. If you are using native XML data store, you should also be familiar with handling XML data through SQL/XML and XQuery.

How this Book is Structured

The following topics are covered:

Chapter 1, "Export," on page 1
Describes the DB2 export utility, used to move data from DB2 tables into files.

Chapter 2, "Import," on page 35
Describes the DB2 import utility, used to move data from files into DB2 tables or views.

Chapter 3, "Load," on page 101
Describes the DB2 load utility, used to move large volumes of data into DB2 tables.

Chapter 4, "Loading data in a partitioned database environment," on page 217
Describes loading data in a partitioned database environment.

Chapter 5, "Moving Data Between Systems," on page 241
Describes how to use the DB2 export, import, and load utilities to transfer data across platforms, and to and from DRDA host databases. DataPropagator, another method for moving data between databases in an enterprise, is also described.

Appendix A, "How to read the syntax diagrams," on page 277
Explains the conventions used in syntax diagrams.

Appendix B, "Differences between the import and load utility," on page 281
Summarizes the important differences between the DB2 load and import utilities.

Appendix C, "Export/Import/Load Sessions - API Sample Program," on page 283
Includes an API sample program that illustrates how to export data to a file, import data to a table, load data into a table, and check the status of a load operation.

"Export/Import/Load Utility File Formats" on page 293
Describes external file formats supported by the database manager export, import, and load utilities.

Appendix E, "Export/Import/Load utility unicode considerations," on page 341
Discusses unicode consideration when using the export, import and load utilities.

Appendix F, "Bind files used by the export, import and load utilities," on page 345
Lists bind files with their default isolation levels, as well as which utilities use them and for what purpose.

Appendix G, "Warning, error and completion messages," on page 347
Provides information about interpreting messages generated by the database manager when a warning or error condition has been detected.

Chapter 1. Export

This chapter describes the DB2 export utility, which is used to write data from a DB2 database to one or more files stored outside of the database. The exported data can then be imported or loaded into another DB2 database, using the DB2 import or the DB2 load utility, respectively, or it can be imported into another application (for example, a spreadsheet).

The following topics are covered:
- "Export Overview"
- "Privileges, authorities and authorization required to use export" on page 3
- "Exporting data" on page 4
- "Exporting XML data" on page 5
- "Using export with identity columns" on page 9
- "Recreating an exported table" on page 9
- "Exporting large objects (LOBS)" on page 10
- "EXPORT " on page 11
- "EXPORT command using the ADMIN_CMD procedure" on page 15
- "db2Export - Export data from a database" on page 19
- "File type modifiers for the export utility" on page 27
- "Export Sessions - CLP Examples" on page 33

For information about exporting data out of typed tables, see "Moving data between typed tables" on page 260. For information about exporting data from a DRDA server database to a file on the DB2 Connect workstation, and the reverse, see "Moving data with DB2 Connect" on page 245.

Export Overview

The export utility exports data from a database to an operating system file, which can be in one of several external file formats. This operating system file can then be used to move the table data to a different server such as DB2 Universal Database for iSeries™.

The following information is required when exporting data:
- An SQL SELECT statement specifying the data to be exported.
- The path and name of the operating system file that will store the exported data.
- The format of the data in the input file. This format can be IXF, WSF, or DEL.
- When exporting typed tables, you might need to provide the subtable traverse order within the hierarchy. If the IXF format is to be used, the default order is recommended. When specifying the order, recall that the subtables must be traversed in the PRE-ORDER fashion. When exporting typed tables, you cannot provide a SELECT statement directly. Instead, you must specify the target subtable name, and optionally a WHERE clause. The export utility uses this information, along with the traverse order, to generate and execute the required SELECT statement.

You can also specify:

- New column names when exporting to IXF or WSF files. If you do not want to specify new column names, the column names in the existing table or view are used in the exported file.
- Additional options to customize the export operation.
- A message file name. During DB2 database operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter. These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

Changes to previous export behavior introduced in DB2 Version 9.1

Following is a summary of syntax changes and changes to export behavior introduced in DB2 Version 9.1:

- In DB2 Universal Database Version 8 (DB2 UDB Version 8), the exported lob file is named, for example, filename.001, filename.002. The default name used by the export utility for lob files is, for example, db2exp.001, db2exp.002. In DB2 V9.1, the exported lob file has a .lob extension, for example, filename.001.lob, filename.002.lob. The default name is named after the input data file name, for example, <datafile>.001.lob, <datafile>.002.lob. If the input date file is generated in DB2 UDB V8, the DB2 V9.1 import utility can read it correctly.
- In DB2 UDB V8, if the LOBS TO option is not specified, then the default export path is the current working directory. In DB2 V9.1, if the LOBS TO option is not specified, then the default export path is the directory in which the exported data file resides.
- In DB2 V9.1, message SQL3040N is improved. Two errors are returned. SQL3040N is returned for lob file errors and SQL3235N is returned for lob path errors. The invalid file name or path name is shown in the message.
- In DB2 UDB V8, the LOBFILE option can contain a path. As a result, the LOB Location Specifier (LLS) in the exported data file also contains a path name. In DB2 V9.1, the LOBFILE option cannot contain a path. For backward compatibility, if the LLS in the input data file contains a path, the Version 9.1 import utility can read the file and import the lob data correctly.
- In DB2 UDB V8, the import and export utilities fail if both the LOBSINFILE and CODEPAGE modifiers are specified together. In DB2 V9.1, both modifiers can be specified together.
- In DB2 UDB V8, if LOBSINFILE is specified and LOBS TO is specified, the specified directory is used for LOB data. Otherwise, LOB data is placed in current working directory. In DB2 V9.1, if LOBSINFILE is specified and LOBS TO is specified, the specified directory is used for LOB data. Otherwise, LOB data is placed in data file directory.
- In DB2 UDB V8, if LOBSINFILE is not specified, then the specified LOBS TO and LOBFILE are ignored. In DB2 V9.1, specifying LOBS TO or LOBFILE implies LOBSINFILE.
- In DB2 UDB V8, export data and message files are created with the default mask on Unix platforms. On Windows platforms, files are created with full permissions if Extended Security is not enabled, and they are created with administrators group full permission and owner read permission if Extended Security is enabled. In DB2 V9.1 export data and message files are created with the user specified umask on unix platforms. On Windows platforms, parent

directory attributes are inherited if Extended Security is not enabled . If Extended Security is enabled, the administrators group has full permission, while the DB2USERS group has read and execute permissions.

- In DB2 V9.1, SQL27984W is returned for the following export scenarios:
  - index column names contain hexadecimal values of 0x2B or 0x2D
  - table contains XML columns
  - table is multidimensional clustered
  - table contains a table partitioning key
  - index name that is longer than 128 bytes due to codepage conversion
  - table is a protected table
  - contains action strings other than SELECT * FROM <TABLE-NAME>
  - method N is specified

  ```
  SQL27984W Some information has not been saved to the PC/IXF file
  during Export. This file will not be supported in
  Import CREATE mode. Reason code="<reason-code>".
  ```

Related concepts:

- "Examples of db2batch tests" in *Performance Guide*
- "Exporting large objects (LOBS)" on page 10
- "Moving data between typed tables" on page 260
- "Privileges, authorities and authorization required to use export" on page 3
- "Recreating an exported table" on page 9
- "Using export with identity columns" on page 9

Related tasks:

- "Exporting data" on page 4

Related reference:

- "Export Sessions - CLP Examples" on page 33
- "Export/Import/Load Utility File Formats" on page 293
- "EXPORT " on page 11

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 19
- "EXPORT " on page 11

Exporting data

The export utility exports data from a database to one of several external file formats. You can specify the data to be exported by supplying an SQL SELECT statement, or by providing hierarchical information for typed tables.

Authorization:

One of the following authorities is required to export data from a database:
- *sysadm*
- *dbadm*

or CONTROL or SELECT privilege on each participating table or view.
- When exporting data from a table with protected rows, only those rows that the session authorization ID is allowed to read are exported.
- If the select includes any protected columns that the session authorization ID is not allowed to read the export fails and an error (SQLCODE 42512) is returned.

Prerequisites:

Before invoking the export utility, you must be connected (or be able to implicitly connect) to the database from which the data will be exported. 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.

Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by performing either a COMMIT or a ROLLBACK before invoking the export utility. There is no requirement for other user applications accessing the table and using separate connections to disconnect.

Restrictions:

The following restrictions apply to the export utility:
- This utility does not support tables with structured type columns.

Procedure:

The export utility can be invoked through the command line processor (CLP), the Export Table notebook in the Control Centre, or an application programming interface (API), **db2Export**.

The following is an example of the EXPORT command issued through the CLP:

```
db2 export to staff.ixf of ixf select * from userid.staff
```

For complete syntax and usage information, see the EXPORT command.

To open the Export Table notebook:
1. From the Control Center, expand the object tree until you find the Tables or Views folder.
2. Click on the folder you want to work with. Any existing tables or views are displayed in the pane on the right side of the window (the contents pane).

3. Right-click on the table or view you want in the contents pane, and select Export from the pop-up menu. The Export Table notebook opens.

Detailed information about the Export Table notebook is provided through the Control Center online help facility.

Related concepts:

- "Export Overview" on page 1

Related reference:

- "EXPORT " on page 11
- "Export/Import/Load Utility File Formats" on page 293
- "ROLLBACK statement" in *SQL Reference, Volume 2*

Exporting XML data

When exporting XML data, the resulting QDM (XQuery Data Model) instances are written to a file or files separate from the main data file containing exported relational data. This is true even if neither the XMLFILE nor the XML TO option is specified. By default, exported QDM instances are all concatenated to the same XML file. You can use the `XMLINSEPFILES` file type modifier to specify that each QDM instance be written to a separate file.

The destination paths and base names of the exported XML files can be specified with the XML TO and XMLFILE options. By default, exported XML files are written to the path of the exported data file. The default base name for exported XML files is the name of the exported data file, with an appending 3-digit sequence number, and the `.xml` extension.

Examples:

For the following examples, imagine a table USER.T1 containing four columns and two rows:

```
C1 INTEGER
C2 XML
C3 VARCHAR(10)
C4 XML
```

Table 1. USER.T1

C1	C2	C3	C4
2	<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to><from> Me</from><heading>note1</heading> <body>Hello World!</body></note>	'char1'	<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</to><from> Her</from><heading>note2</heading> <body>Hello World!</body></note>
4	NULL	'char2'	<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to><from> Them</from><heading>note3</heading> <body>Hello World!</body></note>

Example 1:

The following command exports the contents of USER.T1 in Delimited ASCII (DEL) format to the file ″/mypath/t1export.del″. Because the XML TO and XMLFILE options are not specified, the XML documents contained in columns C2 and C4 are written to the same path as the main exported file ″/mypath″. The base name for these files is ″t1export.del.xml″. The XMLSAVESCHEMA option indicates that XML schema information is saved during the export procedure.

```
EXPORT TO /mypath/t1export.del OF DEL XMLSAVESCHEMA SELECT * FROM USER.T1
```

The exported file ″/mypath/t1export.del″ contains:

```
2,"<XDS FIL='t1export.del.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='t1export.del.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='t1export.del.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

The exported XML file ″/mypath/t1export.del.001.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him
   </to><from>Her</from><heading>note2</heading><body>Hello World!
   </body></note><?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00">
   <to>Us</to><from>Them</from><heading>note3</heading><body>
   Hello World!</body></note>
```

Example 2:

The following command exports the contents of USER.T1 in DEL format to the file ″t1export.del″. XML documents contained in columns C2 and C4 are written to the path ″/home/user/xmlpath″. The XML files are named with the base name ″xmldocs″, with multiple exported XML documents written to the same XML file. The XMLSAVESCHEMA option indicates that XML schema information is saved during the export procedure.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs XMLSAVESCHEMA SELECT * FROM USER.T1
```

The exported DEL file ″/home/user/t1export.del″ contains:

```
2,"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='144' />","char1",
"<XDS FIL='xmldocs.001.xml' OFF='144' LEN='145' />"
4,,"char2","<XDS FIL='xmldocs.001.xml' OFF='289'
LEN='145' SCH='S1.SCHEMA_A' />"
```

The exported XML file ″/home/user/xmlpath/xmldocs.001.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note><?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00">
   <to>Him</to><from>Her</from><heading>note2</heading><body>
   Hello World!</body></note><?xml version="1.0" encoding="UTF-8" ?>
   <note time="14:00:00"><to>Us</to><from>Them</from><heading>
   note3</heading><body>Hello World!</body></note>
```

Example 3:

The following command is similar to Example 2, except that each exported XML document is written to a separate XML file.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLINSEPFILES XMLSAVESCHEMA
SELECT * FROM USER.T1
```

The exported file ″/mypath/t1export.del″ contains:

```
2,"<XDS FIL='xmldocs.001.xml' />","char1","<XDS FIL='xmldocs.002.xml' />"
4,,"char2","<XDS FIL='xmldocs.004.xml' SCH='S1.SCHEMA_A' />"
```

The exported XML file ″/home/user/xmlpath/xmldocs.001.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="12:00:00"><to>You</to>
   <from>Me</from><heading>note1</heading><body>Hello World!</body>
   </note>
```

The exported XML file ″/home/user/xmlpath/xmldocs.002.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="13:00:00"><to>Him</to>
   <from>Her</from><heading>note2</heading><body>Hello World!</body>
   </note>
```

The exported XML file ″/home/user/xmlpath/xmldocs.004.xml″ contains:

```
<?xml version="1.0" encoding="UTF-8" ?><note time="14:00:00"><to>Us</to>
   <from>Them</from><heading>note3</heading><body>Hello World!</body>
   </note>
```

Example 4:

The following command writes the result of an XQuery to an XML file.

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /home/user/xmlpath
XMLFILE xmldocs MODIFIED BY XMLNODECLARATION select
xmlquery( '$m/note/from/text()' passing by ref c4 as "m" returning sequence)
   from USER.T1
```

The exported DEL file ″/mypath/t1export.del″ contains:

```
"<XDS FIL='xmldocs.001.xml' OFF='0' LEN='3' />"
"<XDS FIL='xmldocs.001.xml' OFF='3' LEN='4' />"
```

The exported XML file ″/home/user/xmlpath/xmldocs.001.xml″ contains:

```
HerThem
```

Note: The result of this particular XQuery does not produce well-formed XML documents. Therefore, the file exported above could not be directly imported into an XML column.

Related concepts:

- "Importing XML data" on page 40
- "XML data specifier" on page 244
- "Native XML data store overview" in *XML Guide*

Related reference:

- "LOB and XML file behavior with regard to import and export" on page 7
- "EXPORT " on page 11

LOB and XML file behavior with regard to import and export

LOB and XML files have certain shared behaviors with regard to the import and export utilities.

When exporting data, if one or more LOB paths are specified with the LOBS TO option, the export utility will cycle between the paths to write each successful LOB value to the appropriate LOB file. Similarly, if one or more XML paths are specified with the XML TO option, the export utility will cycle between the paths to write each successive QDM (XQuery Data Model) instance to the appropriate XML file. By default, LOB values and QDM instances are written to the same path to which the exported relational data is written. Unless the LOBSINSEPFILES or XMLINSEPFILES file type modifier is set, both LOB files and XML files can have multiple values concatenated to the same file.

The LOBFILE option provides a means to specify the base name of the LOB files generated by the export utility. Similarly, the XMLFILE option provides a means to specify the base name of the XML files generated by the export utility. The default LOB file base name is the name of the exported data file, with the extension `.lob`. The default XML file base name is the name of the exported data file, with the extension `.xml`. The full name of the exported LOB file or XML file therefore consists of the base name, followed by a number extension that is padded to three digits, and the extension `.lob` or `.xml`.

When importing data, a LOB Location Specifier (LLS) is compatible with an XML target column, and an XML Data Specifier (XDS) is compatible with a LOB target column. If the LOBS FROM option is not specified, the LOB files to import are assumed to reside in the same path as the input relational data file. Similarly, if the XML FROM option is not specified, the XML files to import are assumed to reside in the same path as the input relational data file.

Example 1:

For the following EXPORT command:

```
EXPORT TO /mypath/t1export.del OF DEL MODIFIED BY LOBSINFILE
SELECT * FROM USER.T1
```

All LOB values are written to the file ″/mypath/t1export.del.001.lob″, and all QDM instances are written to the file ″/mypath/t1export.del.001.xml″.

Example 2:

For the following EXPORT command:

```
EXPORT TO /mypath/t1export.del OF DEL LOBS TO /lob1,/lob2
MODIFIED BY LOBSINFILE SELECT * FROM USER.T1
```

The first LOB value will be written to the file ″/lob1/t1export.del.001.lob″, the second will be written to the file ″/lob2/t1export.del.002.lob″, the third will be appended to ″/lob1/t1export.del.001.lob″, the fourth will be appended to ″/lob2/t1export.del.002.lob″, and so on.

Example 3:

For the following EXPORT command:

```
EXPORT TO /mypath/t1export.del OF DEL XML TO /xml1,/xml2 XMLFILE xmlbase
   MODIFIED BY XMLINSEPFILES SELECT * FROM USER.T1
```

The first QDM instance will be written to the file ″/xml1/xmlbase.001.xml″, the second will be written to the file ″/xml2/xmlbase.002.xml″, the third will be written to ″/xml1/xmlbase.003.xml″, the fourth will be written to ″/xml2/xmlbase.004.xml″, and so on.

Example 4:

For a table ″mytable″ that contains a single XML column, and the following IMPORT command:

```
IMPORT FROM myfile.del of del LOBS FROM /lobpath XML FROM /xmlpath
MODIFIED BY LOBSINFILE XMLCHAR replace into mytable
```

If ″myfile.del″ contains the following data:

```
mylobfile.001.lob.123.456/
```

The import utility will try to import an XML document from the file ″/lobpath/mylobfile.001.lob″, starting at file offset 123, with its length being 456 bytes.

The file ″mylobfile.001.lob″ is assumed to be in the LOB path, as opposed to the XML path, since the value is referred to by a LOB Location Specifier (LLS) instead of an XML Data Specifier (XDS).

The document is assumed to be encoded in the character codepage, since the XMLCHAR file type modifier is specified.

Related concepts:

- “XML data type” in *XML Guide*
- “Exporting XML data” on page 5
- “Importing XML data” on page 40

Using export with identity columns

The export utility can be used to export data from a table containing an identity column. If the SELECT statement specified for the export operation is of the form ″select * from tablename″, and the METHOD option is not used, exporting identity column properties to IXF files is supported. The REPLACE_CREATE and the CREATE options of the IMPORT command can then be used to recreate the table, including its identity column properties. If such an IXF file is created from a table containing an identity column of type GENERATED ALWAYS, the only way that the data file can be successfully imported is to specify the `identityignore` modifier. Otherwise, all rows will be rejected (SQL3550W).

Related concepts:

- “Identity columns” in *Administration Guide: Planning*

Recreating an exported table

A table can be saved by using the export utility and specifying the IXF file format. The saved table (including its indexes) can then be recreated using the import utility.

If the column names specified in the index contain either ′-′ or ′+′ characters, the index information is not collected and warning SQL27984W is returned. The export utility completes and the data exported is not affected. The index information is not saved in the IXF file. If you are recreating the table by using the IMPORT CREATE command, the indexes are not recreated. You must create the indexes separately, using the db2look utility.

During an IMPORT CREATE command, warning SQL27984W is returned when some information has not been saved to the PC/IXF file during the export operation. Some information is not saved to the PC/IXF file in the following situations:

- index column names contain hexadecimal values of 0x2B or 0x2D
- table contains XML columns
- table is multidimensional clustered
- table contains a table partitioning key
- index name that is longer than 128 bytes due to codepage conversion
- table is a protected table
- contains action strings other than SELECT * FROM <TABLE-NAME>
- method N is specified

The export operation fails if the data you are exporting exceeds the space available on the file system on which the exported file is created. In this case, you should limit the amount of data selected by specifying conditions on the WHERE clause, so that the export file fits on the target file system. You can invoke the export utility multiple times to export all of the data.

The DEL and ASC file formats do not contain descriptions of the target table, but they do contain the record data. To recreate a table with data stored in these file formats, create the target table, and then use the load, or import utility to populate the table from these files. The db2look utility can be used to capture the original table definitions, and to generate the corresponding data definition language (DDL).

Related concepts:

- "Export Overview" on page 1
- "Import Overview" on page 35
- "Using import to recreate an exported table" on page 45

Related reference:

- "db2look - DB2 statistics and DDL extraction tool command" in *Command Reference*
- "Export/Import/Load Utility File Formats" on page 293
- "EXPORT " on page 11
- "IMPORT " on page 49

Exporting large objects (LOBS)

When exporting data from large object (LOB) columns, the default action is to select the first 32KB of data, and to place this data in the same file as the rest of the column data.

Note: The IXF file format does not store the LOB options of the column, such as whether or not the LOB column is logged. This means that the import utility cannot recreate a table containing a LOB column that is defined to be 1GB or larger.

A LOB Location Specifier (LLS) is used to store multiple LOBs in a single file when exporting LOB information. When exporting data using the `lobsinfile` modifier, the export utility selects the entire LOB file and places it in one of the LOB files.

There might be multiple LOBs per LOB file and multiple LOB files in each LOB path. The data file will contain the LLS records. Use the `lobsinsepfiles` file type modifier to write each LOB into separate file.

An LLS is a string indicating where LOB data can be found within a file. The format of the LLS is `filename.ext.nnn.mmm/`, where `filename.ext` is the name of the file that contains the LOB, `nnn` is the offset of the LOB within the file (measured in bytes), and `mmm` is the length of the LOB (in bytes). For example, an LLS of `db2exp.001.123.456/` indicates that the LOB is located in the file db2exp.001, begins at an offset of 123 bytes into the file, and is 456 bytes long. If the indicated size in the LLS is 0, the LOB is considered to have a length of 0. If the length is -1, the LOB is considered to be NULL and the offset and file name are ignored.

Related reference:

- "EXPORT " on page 11
- "Large objects (LOBs)" in *SQL Reference, Volume 1*
- "db2Export - Export data from a database" on page 19

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:

```
>>--EXPORT TO--filename--OF--filetype--+-------------------------+------->
                                       |          .-,--------.   |
                                       |          V          |   |
                                       '-LOBS TO----lob-path-+---'

>--+---------------------------+--+--------------------------+----------->
   |           .-,--------.    |  |          .-,--------.    |
   |           V          |    |  |          V          |    |
   '-LOBFILE-----filename-+----'  '-XML TO-----xml-path-+----'

>--+---------------------------+--+-------------------------------+------>
   |           .-,--------.    |  |               .-------------. |
   |           V          |    |  |               V             | |
   '-XMLFILE-----filename-+----'  '-MODIFIED BY-----filetype-mod-+-'
```

```
►──┬──────────────┬──┬─────────────────────────────────────┬──────────────────────────►
   └─XMLSAVESCHEMA─┘  │              ┌─,───────────┐        │
                      │              ▼             │        │
                      └─METHOD N──(───column-name──┴──)─────┘

►──┬─select-statement──────────────────────────────────────────────┬─────────────────►◄
   ├─XQUERY──xquery-statement──────────────────────────────────────┤
   └─HIERARCHY──┬─STARTING──sub-table-name─┬──┬──────────────┬─────┘
                └─┤ traversal-order-list ├─┘  └─where-clause─┘
```

traversal-order-list:

```
       ┌─,───────────────┐
       ▼                 │
├──(───sub-table-name────┴──)──────────────────────────────────────────────────────────┤
```

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" on page 1
- "Privileges, authorities and authorization required to use export" on page 3

Related tasks:

- "Exporting data" on page 4

Related reference:

- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "EXPORT command using the ADMIN_CMD procedure" on page 15
- "Export Sessions - CLP Examples" on page 33
- "LOB and XML file behavior with regard to import and export" on page 7

EXPORT command using the ADMIN_CMD procedure

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:

```
>>--EXPORT TO--filename--OF--filetype--+-------------------------+---------->
                                       |            .-,--------. |
                                       |            V          | |
                                       '-LOBS TO------lob-path-+-'
```

```
►─┬──────────────────────────┬─┬───────────────────────┬──────────────────────────────►
  │            ┌─,────────┐  │ │          ┌─,──────┐   │
  │            ▼          │  │ │          ▼        │   │
  └─LOBFILE────filename───┴──┘ └─XML TO───xml-path─┴───┘

►─┬──────────────────────────┬─┬──────────────────────────────────┬────────────────────►
  │            ┌─,────────┐  │ │               ┌──────────────┐   │
  │            ▼          │  │ │               ▼              │   │
  └─XMLFILE────filename───┴──┘ └─MODIFIED BY───filetype-mod───┴───┘

►─┬───────────────┬─┬───────────────────────────────────────┬───────────────────────────►
  └─XMLSAVESCHEMA─┘ │                 ┌─,─────────────┐     │
                    │                 ▼               │     │
                    └─METHOD N──(─────column-name─────┴──)──┘

►──┬─select-statement───────────────────────────────────────────┬─────────────────────►◄
   ├─XQUERY──xquery-statement───────────────────────────────────┤
   └─HIERARCHY──┬─STARTING──sub-table-name──┬──┬──────────────┬─┘
                └─┤ traversal-order-list ├──┘  └─where-clause─┘
```

traversal-order-list:

```
       ┌─,──────────────────┐
       ▼                    │
├──(───sub-table-name───────┴──)──────────────────────────────────────────────────┤
```

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:

- "Privileges, authorities and authorization required to use export" on page 3

Related reference:

- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "ADMIN_GET_MSGS table function – Retrieve messages generated by a data movement utility that is executed through the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "ADMIN_REMOVE_MSGS procedure – Clean up messages generated by a data movement utility that is executed through the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "db2Export - Export data from a database" on page 19
- "Miscellaneous variables" in *Performance Guide*
- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*

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" on page 4

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 – Run administrative commands" in *Administrative SQL Routines and Views*
- "EXPORT " on page 11
- "EXPORT command using the ADMIN_CMD procedure" on page 15
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 73
- "db2Load - Load data into a table" on page 161

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++)"

File type modifiers for the export utility

Table 2. Valid file type modifiers for the export utility: All file formats

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB 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. If you specify the "lobsinfile" modifier when using EXPORT, the LOB data is placed in the locations specified by the LOBS TO clause. Otherwise the LOB data is sent to the data file directory. The LOBS TO clause 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 LOBS TO or LOBFILE options will implicitly activate the LOBSINFILE behavior. 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/.
xmlinsepfiles	Each XQuery Data Model (QDM) instance is written to a separate file. By default, multiple values are concatenated together in the same file.
lobsinsepfiles	Each LOB value is written to a separate file. By default, multiple values are concatenated together in the same file.
xmlnodeclaration	QDM instances are written without an XML declaration tag. By default, QDM instances are exported with an XML declaration tag at the beginning that includes an encoding attribute.

File type modifiers for the export utility

Table 2. Valid file type modifiers for the export utility: All file formats (continued)

Modifier	Description
xmlchar	QDM instances are written in the character codepage. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, QDM instances are written out in Unicode.
xmlgraphic	If the `xmlgraphic` modifier is specified with the EXPORT command, the exported XML document will be encoded in the UTF-16 code page regardless of the application code page or the `codepage` file type modifier.

Table 3. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format

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.[2] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	The single quotation mark (') can also be specified as a character string delimiter as follows:
	`modified by chardel''`

codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

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.[2]
	In the following example, `coldel;` causes the export utility to use the semicolon character (;) as a column delimiter for the exported data:
	`db2 "export to temp of del modified by coldel;`
	`   select * from staff where dept = 20"`

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.[2]
nochardel	Column data will not be surrounded by character delimiters. This option should not be specified if the data is intended to be imported or loaded using DB2. 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` or `nodoubledel`. These are mutually exclusive options.
nodoubledel	Suppresses recognition of double character delimiters.[2]

Table 3. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

striplzeros	Removes the leading zeros from all exported decimal columns.
	
	Consider the following example:
	`db2 create table decimalTable ( c1 decimal( 31, 2 ) )`
	`db2 insert into decimalTable values ( 1.1 )`
	
	`db2 export to data of del select * from decimalTable`
	
	`db2 export to data of del modified by STRIPLZEROS`
	`   select * from decimalTable`
	
	In the first export operation, the content of the exported file data will be +00000000000000000000000000001.10. In the second operation, which is identical to the first except for the `striplzeros` modifier, the content of the exported file data will be +1.10.

File type modifiers for the export utility

Table 3. Valid file type modifiers for the export utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[4] 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)`
	
	Following is an example of a time stamp format:
	`"YYYY/MM/DD HH:MM:SS.UUUUUU"`
	
	The MMM element will produce the following values: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', and 'Dec'. 'Jan' is equal to month 1, and 'Dec' is equal to month 12.
	
	The following example illustrates how to export data containing user-defined time stamp formats from a table called 'schedule':
	`db2 export to delfile2 of del`
	`   modified by timestampformat="yyyy.mm.dd hh:mm tt"`
	`   select * from schedule`

Table 4. Valid file type modifiers for the export utility: IXF file format

Modifier	Description
codepage=*x*	*x* is an ASCII character string. The value is interpreted as the code page of the data in the output data set. Converts character data from this code page to the application code page during the export operation. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. The `codepage` modifier cannot be used with the `lobsinfile` modifier.

Table 5. Valid file type modifiers for the export utility: WSF file format

Modifier	Description
1	Creates a WSF file that is compatible with Lotus 1-2-3 Release 1, or Lotus 1-2-3 Release 1a.[5] This is the default.
2	Creates a WSF file that is compatible with Lotus Symphony Release 1.0.[5]
3	Creates a WSF file that is compatible with Lotus 1-2-3 Version 2, or Lotus Symphony Release 1.1.[5]
4	Creates a WSF file containing DBCS characters.

Notes:

1. The export 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 export operation fails, and an error code is returned.
2. Delimiter restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.
3. The export utility normally writes
   - date data in *YYYYMMDD* format
   - char(date) data in "*YYYY-MM-DD*" format
   - time data in "*HH.MM.SS*" format
   - time stamp data in "*YYYY-MM-DD-HH. MM.SS.uuuuuu*" format

   Data contained in any datetime columns specified in the SELECT statement for the export operation will also be in these formats.
4. 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)
   ```
5. These files can also be directed to a specific product by specifying an `L` for Lotus 1-2-3, or an `S` for Symphony in the *filetype-mod* parameter string. Only one value or product designator can be specified.

6. The WSF file format is not supported for XML columns.
7. All QDM instances are written to XML files that are separate from the main data file, even if neither the ″XMLFILE″ nor the ″XML TO″ clause is specified. By default, XML files are written to the path of the exported data file. The default base name for XML files is the name of the exported data file with the ″.xml″ appended to it.
8. All QDM instances are written with an XML declaration at the beginning that includes an encoding attribute, unless the XMLNODECLARATION file type modifier is specified.
9. By default, all QDM instances are written in Unicode unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. The default path for XML data and LOB data is the path of the main data file. The default XML file base name is the main data file. The default LOB file base name is the main data file. For example, if the main data file is

    ```
    /mypath/myfile.del
    ```

    , the default path for XML data and LOB data is

    ```
    /mypath"
    ```

    , the default XML file base name is

    ```
    myfile.del
    ```

    , and the default LOB file base name is

    ```
    myfile.del
    ```

    .

    The LOBSINFILE file type modifier must be specified in order to have LOB files generated.
11. The export utility appends a numeric identifier to each LOB file or XML file. The identifier is a 3 digit, 0 padded sequence value, starting at

    ```
    .001
    ```

    . After the 999th LOB file or XML file, the identifier will no longer be padded with zeroes (for example, the 1000th LOG file or XML file will have an extension of

    ```
    .1000
    ```

    . Following the numeric identifier is a three character type identifier representing the data type, either

    ```
    .lob
    ```

    or

    ```
    .xml
    ```

    . For example, a generated LOB file would have a name in the format

    ```
    myfile.del.001.lob
    ```

    , and a generated XML file would be have a name in the format

    ```
    myfile.del.001.xml
    ```

    .
12. It is possible to have the export utility export QDM instances that are not well-formed documents by specifying an XQuery. However, you will not be

able to import or load these exported documents directly into an XML column, since XML columns can only contain complete documents.

Related reference:

- "Delimiter restrictions for moving data" on page 259
- "db2Export - Export data from a database" on page 19
- "EXPORT " on page 11

Export Sessions - CLP Examples

The following example shows how to export information from the STAFF table in the SAMPLE database (to which the user must be connected) to `myfile.ixf`, with the output in IXF format. If the database connection is not through DB2 Connect, the index definitions (if any) will be stored in the output file; otherwise, only the data will be stored:

```
db2 export to myfile.ixf of ixf messages msgs.txt select * from staff
```

The following example shows how to export the information about employees in Department 20 from the STAFF table in the SAMPLE database (to which the user must be connected) to `awards.ixf`, with the output in IXF format:

```
db2 export to awards.ixf of ixf messages msgs.txt select * from staff
   where dept = 20
```

The following example shows how to export LOBs to a DEL file:

```
db2 export to myfile.del of del lobs to mylobs/
   lobfile lobs1, lobs2 modified by lobsinfile
   select * from emp_photo
```

The following example shows how to export LOBs to a DEL file, specifying a second directory for files that might not fit into the first directory:

```
db2 export to myfile.del of del
   lobs to /db2exp1/, /db2exp2/ modified by lobsinfile
   select * from emp_photo
```

The following example shows how to export data to a DEL file, using a single quotation mark as the string delimiter, a semicolon as the column delimiter, and a comma as the decimal point. The same convention should be used when importing data back into the database:

```
db2 export to myfile.del of del
   modified by chardel'' coldel; decpt,
   select * from staff
```

Related concepts:

- "Export Overview" on page 1

Related tasks:

- "Exporting data" on page 4

Related reference:

- "EXPORT " on page 11

Chapter 2. Import

This chapter describes the DB2 import utility, which uses the SQL INSERT statement to write data from an input file into a table or view. If the target table or view already contains data, you can either replace or append to the existing data.

The following topics are covered:

For information about importing data from typed tables, see "Moving data between typed tables" on page 260. For information about importing data from a file on the DB2 Connect workstation to a DRDA server database, and the reverse, see "Moving data with DB2 Connect" on page 245.

Import Overview

The import utility inserts data from an input file into a table or updatable view. If the table or view receiving the imported data already contains data, you can either replace or append to the existing data.

The following information is required when importing data:

- The path and the name of the input file.
- The name or alias of the target table or view.
- The format of the data in the input file. This format can be IXF, WSF, DEL, or ASC.
- Whether the input data is to be inserted into the table or view, or whether existing data in the table or view is to be updated or replaced by the input data.
- A message file name, if the utility is invoked through the application programming interface (API), **db2Import**.

- When working with typed tables, you might need to provide the method or order by which to progress through all of the structured types. The order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy is called the *traverse* order. This order is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.

  When working with typed tables, you might also need to provide the subtable list. This list shows into which subtables and attributes to import data.

You can also specify:

- The method to use for importing the data: column location, column name, or relative column position.
- The number of rows to INSERT before committing the changes to the table. Requesting periodic COMMITs reduces the number of rows that are lost if a failure and a ROLLBACK occur during the import operation. It also prevents the DB2® logs from getting full when processing a large input file.
- The number of file records to skip before beginning the import operation. If an error occurs, you can restart the import operation immediately following the last row that was successfully imported and committed.
- The names of the columns within the table or view into which the data is to be inserted.
- A message file name. During DB2 operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter. These message files are standard ASCII text files. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

Note: Specifying target table column names or a specific importing method makes importing to a remote database slower.

Changes to previous import behavior introduced in DB2 Version 9.1

The following is a summary of changes introduced in DB2 Version 9.1:

- In DB2 UDB V8, if a lob file is not found, the row is rejected if the column is not nullable, or NULL if the column is nullable. In DB2 V9.1, if a lob file is not found, the row is rejected regardless of the nullability of the column.
- In DB2 UDB V8, if the ixf file codepage is different from the application codepage, the import utility returns SQL3050W. In DB2 V9, SQL3050W is not returned. In DB2 V9.1, message SQL3040N is improved, to return two separate errors. SQL3040N is returned for lobfile errors and SQL3235N is returned for lob path errors. The invalid file name or path name is indicated in the message.
- In DB2 UDB V8, if the LOB Location Specifier (LLS) contains a path, for example, the LLS is /home/try/newlob.001.12.345/ and the path is invalid, SQL3040N reason code 6 is returned and the utility exits immediately. In DB2 V9.1, the row is rejected and processing continues. In DB2 V9.1, the exported LLS never contains a path name.
- In DB2 UDB V8, the import and export utilities fail if both the LOBSINFILE and CODEPAGE modifiers are specified together. In DB2 V9.1, both modifiers can be specified together.

- In DB2 UDB V8, if LOBSINFILE is not specified, then the specified LOBS FROM is ignored. In DB2 V9.1, specifying LOBS FROM implies LOBSINFILE.
- In DB2 UDB V8, if LOBSINFILE is specified, and LOBS FROM is specified, the specified lob directory is searched first, then the current working directory. If LOBS FROM is not specified, the current working directory is searched. In DB2 V9.1, if LOBSINFILE is specified, and LOBS FROM is specified, the specified lob directory is searched first, then the current working directory. If LOBS FROM is not specified, the data file directory is search first, then the current working directory.
- In DB2 UDB V8, import data and message files are created with the default mask on unix platforms. On Windows platforms, files are created with full permissions if Extended Security is not enabled and with administrators group full permission and owner read permission if Extended Security is enabled. In DB2 V9.1, import data and message files are created with the user specified umask on unix platforms. On Windows platforms, parent directory attributes are inherited if Extended Security is not enabled . If Extended Security is enabled, the administrators group has full permission, while the DB2USERS group has read and execute permissions.
- During IMPORT CREATE from an IXF file exported in DB2 Version 9.1, the following error is returned:

  ```
  SQL3311N This PC/IXF file is not supported in Import CREATE
  mode. Reason code ="<reason-code>".
  ```

  The possible causes include:
  - the index column names contain hexadecimal values of 0x2B or 0x2D
  - file is exported from a table containing XML columns
  - file was exported from an MDC table
  - file was exported from table with a table partitioning key
  - index name contains more than 128 bytes after codepage conversion
  - file was exported from a protected table
  - action string other than SELECT * FROM <TABLE-NAME> was used during the export operation
  - method N was used during the export operation

  This file cannot be used in IMPORT CREATE operations to recreate the table because some information is missing. For reason codes 1, 3, 4, 5, 7 and 8, you can use the file type modifier FORCECREATE to force the CREATE operation with this file. For reason codes 2 and 6, you can use the db2look tool to extract table information and perform IMPORT INSERT or REPLACE operation.

Related concepts:
- “Moving data between typed tables” on page 260

Related tasks:
- “Importing data” on page 38

Related reference:
- “Export/Import/Load Utility File Formats” on page 293
- “Import sessions - CLP examples” on page 97
- “IMPORT ” on page 49

Privileges, authorities, and authorization required to use import

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.

To use the import utility to create a new table, you must have SYSADM authority, DBADM authority, or CREATETAB privilege for the database. To replace data in an existing table or view, you must have SYSADM authority, DBADM authority, or CONTROL privilege for the table or view, or INSERT, SELECT, UPDATE and DELETE privileges for the table or view. To append data to an existing table or view, you must have SELECT and INSERT privileges for the table or view. To use the REPLACE or REPLACE_CREATE option on a table, the session authorization ID must have the authority to drop the table.

Notes:

- 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.
- To import 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:

- "IMPORT " on page 49
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 73

Importing data

The import utility inserts data from an external file with a supported file format into a table, hierarchy, view or nickname. The load utility is a faster alternative, but the load utility does not support loading data at the hierarchy level.

Prerequisites:

Before invoking the import utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be imported. If implicit connect is enabled, a connection to the default database is established. Utility access to DB2 for Linux, UNIX, or Windows database servers from DB2 for Linux, UNIX, or Windows clients must be a direct connection through the engine and not through a DB2 Connect gateway or loop back environment. Since the utility will issue a COMMIT or a ROLLBACK statement, you should complete all transactions and release all locks by issuing a COMMIT statement or a ROLLBACK operation before invoking import.

Restrictions:

The following restrictions apply to the import utility:

- If the existing table is a parent table containing a primary key that is referenced by a foreign key in a dependent table, its data cannot be replaced, only appended to.
- You cannot perform an import replace operation into an underlying table of a materialized query table defined in refresh immediate mode.
- You cannot import data into a system table, a summary table, or a table with a structured type column.
- You cannot import data into declared temporary tables.
- Views cannot be created through the import utility.
- 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 *`.)
- Because the import utility generates its own SQL statements, the maximum statement size of 2MB might, in some cases, be exceeded.
- You cannot recreate a partitioned table or an multidimensional clustered table (MDC) using the CREATE or REPLACE_CREATE import options.
- You cannot recreate tables containing XML columns.

The following limitation applies to the import utility:

If the volume of output messages generated by an import operation against a remote database exceeds 60KB, the utility will keep the first 30KB and the last 30KB.

Procedure:

The import utility can be invoked through the command line processor (CLP), the Import Table notebook in the Control Centre, or by calling an application programming interface (API), **db2Import** from a client application.

Following is an example of the IMPORT command issued through the CLP:

```
db2 import from stafftab.ixf of ixf insert into userid.staff
```

To open the Import Table notebook:

1. From the Control Center, expand the object tree until you find the Tables folder.
2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane).
3. Right-click on the table you want in the contents pane, and select Import from the pop-up menu. The Import Table notebook opens.

Detailed information about the Import Table notebook is provided through the Control Center online help facility.

Related concepts:

- "Import Overview" on page 35
- "Importing large objects (LOBS)" on page 46

Related reference:

- "ROLLBACK statement" in *SQL Reference, Volume 2*
- "Import sessions - CLP examples" on page 97
- "IMPORT " on page 49

Importing XML data

When importing data into an XML table column, you can use the XML FROM option to specify the paths of the input XML data file or files. For example, For an XML file ″/home/user/xmlpath/xmldocs.001.xml″ that had previously been exported, the following command could be used to import the data back into the table.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath INSERT INTO USER.T1
```

Validating Inserted Documents Against Schemas:

The XMLVALIDATE option allows XML documents to be validated against XML schemas as they are imported. In the following example, incoming XML documents are validated against schema information that was saved when the XML documents were exported:

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLVALIDATE
     USING XDS INSERT INTO USER.T1
```

Specifying Parse Options:

You can use the XMLPARSE option to specify whether whitespace in the imported XML documents is preserved or stripped. In the following example, all imported XML documents are validated against XML schema information that was saved when the XML documents were exported, and these documents are parsed with whitespace preserved.

```
IMPORT FROM t1export.del OF DEL XML FROM /home/user/xmlpath XMLPARSE PRESERVE
WHITESPACE XMLVALIDATE USING XDS INSERT INTO USER.T1
```

Related concepts:

- "Exporting XML data" on page 5
- "Native XML data store overview" in *XML Guide*

Related reference:

- "LOB and XML file behavior with regard to import and export" on page 7
- "IMPORT " on page 49

Using import in a client/server environment

When you import a file to a remote database, a stored procedure can be called to perform the import on the server. A stored procedure will not be called when:

- The application and database code pages are different.
- The file being imported is a multiple-part PC/IXF file.
- The method used for importing the data is either column name or relative column position.
- The target column list provided is longer than 4KB.
- The LOBS FROM clause or the `lobsinfile` modifier is specified.
- The NULL INDICATORS clause is specified for ASC files.

When import uses a stored procedure, messages are created in the message file using the default language installed on the server. The messages are in the language of the application if the language at the client and the server are the same.

The import utility creates two temporary files in the `tmp` subdirectory of the `sqllib` directory (or the directory indicated by the **DB2INSTPROF** registry variable, if specified). One file is for data, and the other file is for messages generated by the import utility.

If you receive an error about writing or opening data on the server, ensure that:

- The directory exists.
- There is sufficient disk space for the files.
- The instance owner has write permission in the directory.

Related concepts:

- "Import Overview" on page 35

Using import with buffered inserts

In a partitioned database environment, the import utility can be enabled to use buffered inserts. This reduces the messaging that occurs when data is imported, resulting in better performance; however, since details about a failed buffered insert are not returned, this option should only be enabled if you are not concerned about error reporting.

When buffered inserts are used, import sets a default WARNINGCOUNT value to *1*. As a result, the utility will fail if any rows are rejected. If a record is rejected, the utility will roll back the current transaction. The number of committed records can be used to determine which records were successfully inserted into the database. The number of committed records can be non zero only if the COMMITCOUNT option was specified.

If a different WARNINGCOUNT value is explicitly specified on the import command, and some rows were rejected, the row summary output by the utility can be incorrect. This is due to a combination of the asynchronous error reporting used with buffered inserts and the fact that an error detected during the insertion of a group of rows causes all the rows of that group to be backed out. Since the utility would not reliably report which input records were rejected, it would be difficult to determine which records were committed and which records need to be re-inserted into the database.

Use the DB2 bind utility to request buffered inserts. The import package, `db2uimpm.bnd`, must be rebound against the database using the INSERT BUF option. For example:

```
db2 connect to your_database
db2 bind db2uimpm.bnd insert buf
```

Buffered inserts feature cannot be used in conjunction with import operations in which the INSERT_UPDATE parameter is specified. Bind file db2uImpInsUpdate.bnd enforces this restriction. This file should never be bound with the INSERT BUF option. This causes the import operations in which the INSERT_UPDATE parameter is specified, to fail. Import operations in which the INSERT, REPLACE or REPLACE_CREATE parameter is specified are not affected by the binding of the new file.

Related concepts:

- "Import Overview" on page 35

Using import with identity columns

The import utility can be used to import data into a table containing an identity column. If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the import utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The import utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values will not be reported. In addition, the `compound=x` modifier cannot be used when importing data into a table with an identity column.

Two file type modifiers are supported by the import utility to simplify its use with tables that contain an identity column:

- The `identitymissing` modifier makes importing a table with an identity column more convenient if the input data file does not contain any values (not even NULLS) for the identity column. For example, consider a table defined with the following SQL statement:

  ```
  create table table1 (c1 char(30),
                       c2 int generated by default as identity,
                       c3 real,
                       c4 char(1))
  ```

  A user might want to import data from a file (`import.del`) into TABLE1, and this data might have been exported from a table that does not have an identity column. The following is an example of such a file:

  ```
  Robert, 45.2, J
  Mike, 76.9, K
  Leo, 23.4, I
  ```

  One way to import this file would be to explicitly list the columns to be imported through the IMPORT command as follows:

  ```
  db2 import from import.del of del replace into table1 (c1, c3, c4)
  ```

  For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of importing the file is to use the `identitymissing` file type modifier as follows:

  ```
  db2 import from import.del of del modified by identitymissing
     replace into table1
  ```

- The `identityignore` modifier is in some ways the opposite of the `identitymissing` modifier: it indicates to the import utility that even though the input data file contains data for the identity column, the data should be ignored, and an identity value should be generated for each row. For example, a user might want to import the following data from a file (`import.del`) into TABLE1, as defined above:

  ```
  Robert, 1, 45.2, J
  Mike, 2, 76.9, K
  Leo, 3, 23.4, I
  ```

If the user-supplied values of 1, 2, and 3 are not to be used for the identity column, the user could issue the following IMPORT command:

```
db2 import from import.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The `identityignore` modifier simplifies the syntax as follows:

```
db2 import from import.del of del modified by identityignore
   replace into table1
```

When a table with an identity column is exported to an IXF file, the REPLACE_CREATE and the CREATE options of the IMPORT command can be used to recreate the table, including its identity column properties. If such an IXF file is created from a table containing an identity column of type GENERATED ALWAYS, the only way that the data file can be successfully imported is to specify the `identityignore` modifier. Otherwise, all rows will be rejected (SQL3550W).

Related concepts:

- "Identity columns" in *Administration Guide: Planning*

Using import with generated columns

The import utility can be used to import data into a table containing (non-identity) generated columns.

If no generated column-related file type modifiers are used, the import utility works according to the following rules:

- A value will be generated for a generated column whenever the corresponding row in the input file is missing a value for the column, or a NULL value is explicitly given. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).
- If the server generates a NULL value for a generated column that is not nullable, the row of data to which this field belongs is rejected (SQL0407N). This could happen, for example, if a non-nullable generated column were defined as the sum of two table columns that have NULL values supplied to them in the input file.

Two file type modifiers are supported by the import utility to simplify its use with tables that contain generated columns:

- The `generatedmissing` modifier makes importing data into a table with generated columns more convenient if the input data file does not contain any values (not even NULLS) for all generated columns present in the table. For example, consider a table defined with the following SQL statement:

  ```
  create table table1 (c1 int,
                       c2 int,
                       g1 int generated always as (c1 + c2),
                       g2 int generated always as (2 * c1),
                       c3 char(1))
  ```

  A user might want to import data from a file (`load.del`) into TABLE1, and this data might have been exported from a table that does not have any generated columns. The following is an example of such a file:

  ```
  1, 5, J
  2, 6, K
  3, 7, I
  ```

One way to import this file would be to explicitly list the columns to be imported through the IMPORT command as follows:

```
db2 import from import.del of del replace into table1 (c1, c2, c3)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of importing the file is to use the `generatedmissing` file type modifier as follows:

```
db2 import from import.del of del modified by generatedmissing
   replace into table1
```

- The `generatedignore` modifier is in some ways the opposite of the `generatedmissing` modifier: it indicates to the import utility that even though the input data file contains data for all generated columns, the data should be ignored, and values should be generated for each row. For example, a user might want to import the following data from a file (`import.del`) into TABLE1, as defined above:

  ```
  1, 5, 10, 15, J
  2, 6, 11, 16, K
  3, 7, 12, 17, I
  ```

  The user-supplied, non-NULL values of `10`, `11`, and `12` (for g1), and `15`, `16`, and `17` (for g2) result in the row being rejected (SQL3550W). To avoid this, the user could issue the following IMPORT command:

  ```
  db2 import from import.del of del method P(1, 2, 5)
     replace into table1 (c1, c2, c3)
  ```

  Again, this approach might be cumbersome and prone to error if the table has many columns. The `generatedignore` modifier simplifies the syntax as follows:

  ```
  db2 import from import.del of del modified by generatedignore
     replace into table1
  ```

- When using the INSERT_UPDATE clause, if the generated column is also a primary key and the `generatedignore` modifier is specified, the IMPORT command honours the `generatedignore` modifier. The IMPORT command does not substitute the user supplied value for this column in the WHERE clause of the UPDATE statement.

Related concepts:

- "Generated Columns" in *Developing SQL and External Routines*
- "Import Overview" on page 35
- "Importing large objects (LOBS)" on page 46
- "Using import in a client/server environment" on page 40
- "Using import to recreate an exported table" on page 45
- "Using import with buffered inserts" on page 41
- "Using import with identity columns" on page 42

Related tasks:

- "Importing data" on page 38

Related reference:

- "Import sessions - CLP examples" on page 97
- "IMPORT " on page 49

Using import to recreate an exported table

You can use the import utility to recreate a table that was saved through the export utility. The table must have been exported to an IXF file, and the SELECT statement used during the export operation must have met certain conditions (for example, no column names can be used in the SELECT clause; only `select *` is permitted). When creating a table from an IXF file, not all attributes of the original table are preserved. For example, referential constraints, foreign key definitions, and user-defined data types are not retained. The following attributes of the original table are retained:

- Primary key name, and definition
- Column information:
  - Column name
  - Column data type, including user-defined distinct types, which are preserved as their base type
  - Identity properties
  - Lengths (except for lob_file types)
  - Code page (if applicable)
  - Identity options
  - Whether the column is defined as nullable or not nullable
  - Default values for constants, if any, but not other types of default values
- Index Information (provided the column names in the index do not contain characters '-' or '+'):
  - Index name
  - Index creator name
  - Column names, and whether each column is sorted in ascending, or in descending order
  - Whether the index is defined as unique
  - Whether the index is clustered
  - Whether the index allows reverse scans
  - *pctfree* values
  - *minpctused* values

The following attributes of the original table are *not* retained (This list is not exhaustive, use with care):

- Whether the source was a normal table, a materialized query table, a view, or a set of columns from any or all of these sources
- Unique constraints and other types of constraints or triggers (not including Primary Key constraints).
- Table information:
  - Materialized query table definition (if applicable)
  - Materialized query table options (if applicable)
  - Table space options; however, this information can be specified through the IMPORT command
  - Multidimensional clustering (MDC) dimensions
  - Partitioned table dimensions
  - Table partitioning key
  - Not logged initially property

  - Check constraints
  - Table codepage
  - Protected table properties
  - Table or value compression options
- Column information:
  - Any default value except constant values
  - LOB options (if any)
  - XML properties
  - References clause of the create table statement (if any)
  - Referential constraints (if any)
  - Check constraints (if any)
  - Generated column options (if any)
  - Columns dependent on database scope Sequences
- Index information:
  - Include columns (if any)
  - Index name, if the index is a primary key index
  - Descending order of keys, if the index is a primary key index (Ascending is the default)
  - Index column names contain hexadecimal values of 0x2B or 0x2D
  - Index name contains more than 128 bytes after codepage conversion
  - PCTFREE2 value
  - UNIQUE constraints

Related concepts:
- "Export Overview" on page 1
- "Import Overview" on page 35
- "Recreating an exported table" on page 9

Related reference:
- "EXPORT " on page 11
- "IMPORT " on page 49

Importing large objects (LOBS)

When importing into large object (LOB) columns, the data can come either from the same file as the rest of the column data, or from separate files. If the data is coming from separate files, the LOBSINFILE file type modifier must be specified for DEL, ASC and WSF files.

The column in the main input data file contains either the import data (default), or the name of a file where the import data is stored.

Notes:
1. When LOB data is stored in the main input data file, no more than 32KB of data is allowed. Truncation warnings are ignored.
2. All of the LOB data must be stored in the main file, or each LOB is stored in separate files. The main file cannot have a mixture of LOB data and file names. LOB values are imported from separate files by using the `lobsinfile` modifier, and the `LOBS FROM` clause.

A LOB Location Specifier (LLS) can be used to store multiple LOBs in a single file when importing, exporting and loading LOB information.

An LLS is a string indicating where LOB data can be found within a file. The format of the LLS is `filename.ext.nnn.mmm/`, where `filename.ext` is the name of the file that contains the LOB, `nnn` is the offset of the LOB within the file (measured in bytes), and `mmm` is the length of the LOB (in bytes). For example, an LLS of `db2exp.001.123.456/` indicates that the LOB is located in the file db2exp.001, begins at an offset of 123 bytes into the file, and is 256 bytes long. If the indicated size in the LLS is 0, the LOB is considered to have a length of 0. If the length is -1, the LOB is considered to be NULL and the offset and file name are ignored.

When importing or loading data with the `modified by lobsinfile` option specified, An LLS will be expected for each of the corresponding LOB columns. If something other than an LLS is encountered for a LOB column, the database will treat it as a LOB file, and will load the entire file as the LOB.

Related reference:

- "IMPORT " on page 49
- "Data Type-Specific Rules Governing PC/IXF File Import into Databases" on page 330
- "General Rules Governing PC/IXF File Import into Databases" on page 328
- "Large objects (LOBs)" in *SQL Reference, Volume 1*

Importing user-defined distinct types (UDTs)

The import utility casts user-defined distinct types (UDTs) to similar base data types automatically. This saves you from having to explicitly cast UDTs to the base data types. Casting allows for comparisons between UDTs and the base data types in SQL.

Related concepts:

- "User-defined distinct types" in *Developing SQL and External Routines*

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" on page 203

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:

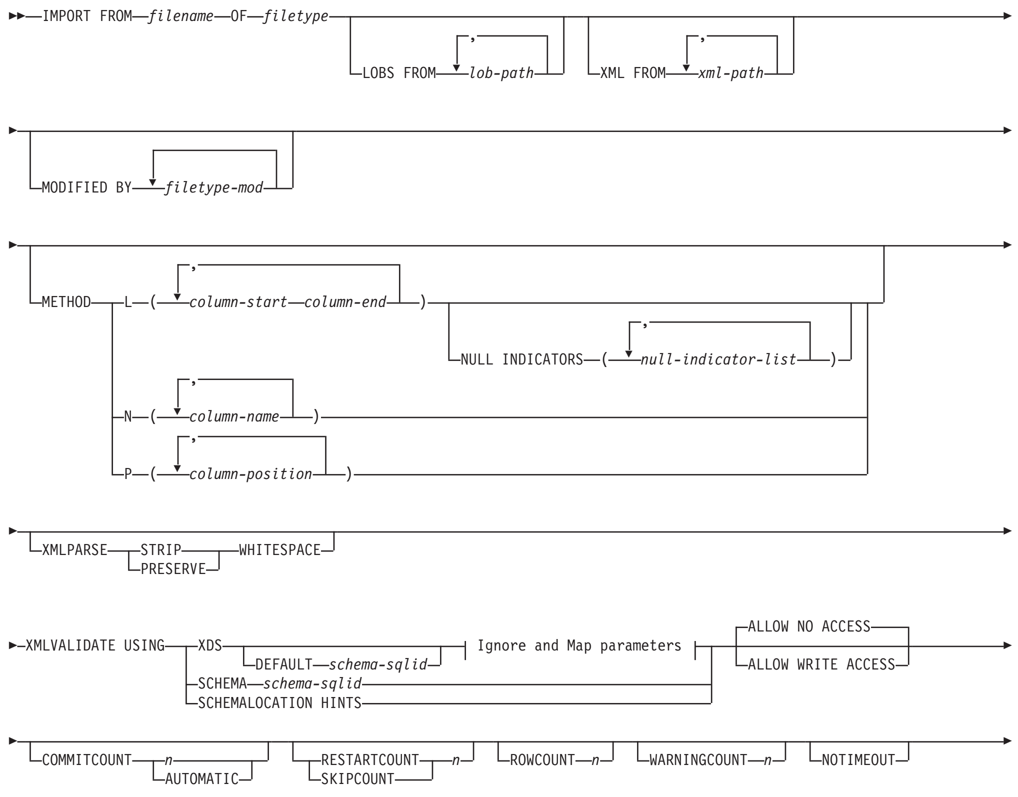
- IMPORT using the INSERT option requires one of the following:
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on each participating table, view, or nickname
  - INSERT and SELECT privilege on each participating table or view
- IMPORT to an existing table using the INSERT_UPDATE option, requires one 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:

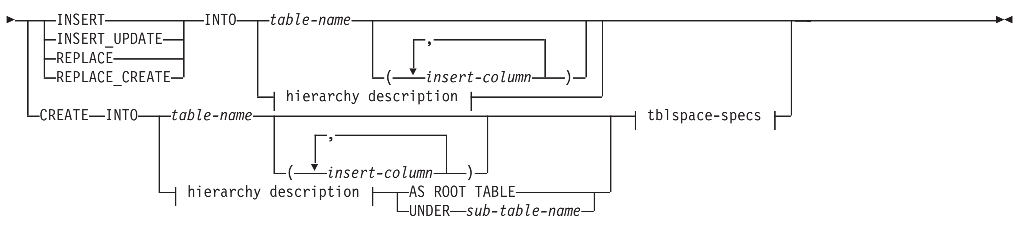
  - *sysadm*
  - *dbadm*
  - CONTROL privilege on every sub-table in the hierarchy
- To import data into a table that has protected columns, the session authorization ID must have LBAC credentials that allow write access to all protected columns in the table. Otherwise the import fails and an error (SQLSTATE 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.

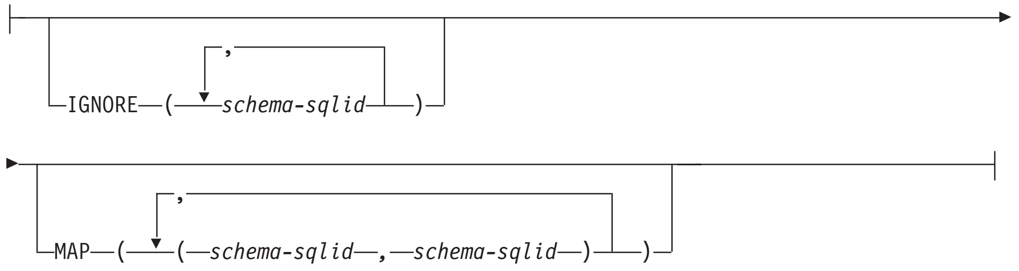
Required connection:

Command syntax:

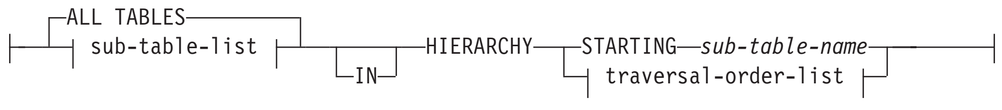




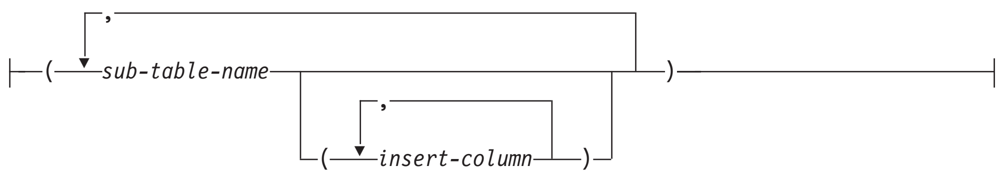
Ignore and Map parameters:



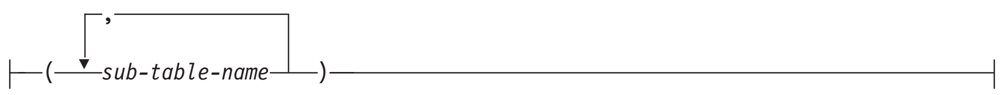
hierarchy description:



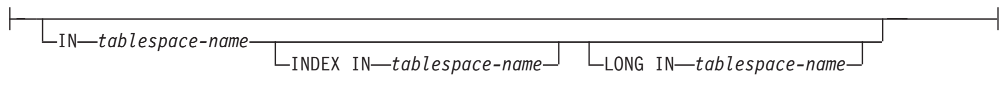
sub-table-list:



traversal-order-list:



tblspace-specs:



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" on page 35
- "Privileges, authorities, and authorization required to use import" on page 38

Related tasks:
- "Importing data" on page 38

Related reference:
- "XMLPARSE scalar function" in *SQL Reference, Volume 1*
- "ADMIN_CMD procedure – 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" on page 61
- "Import sessions - CLP examples" on page 97
- "LOB and XML file behavior with regard to import and export" on page 7

IMPORT command using the ADMIN_CMD procedure

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

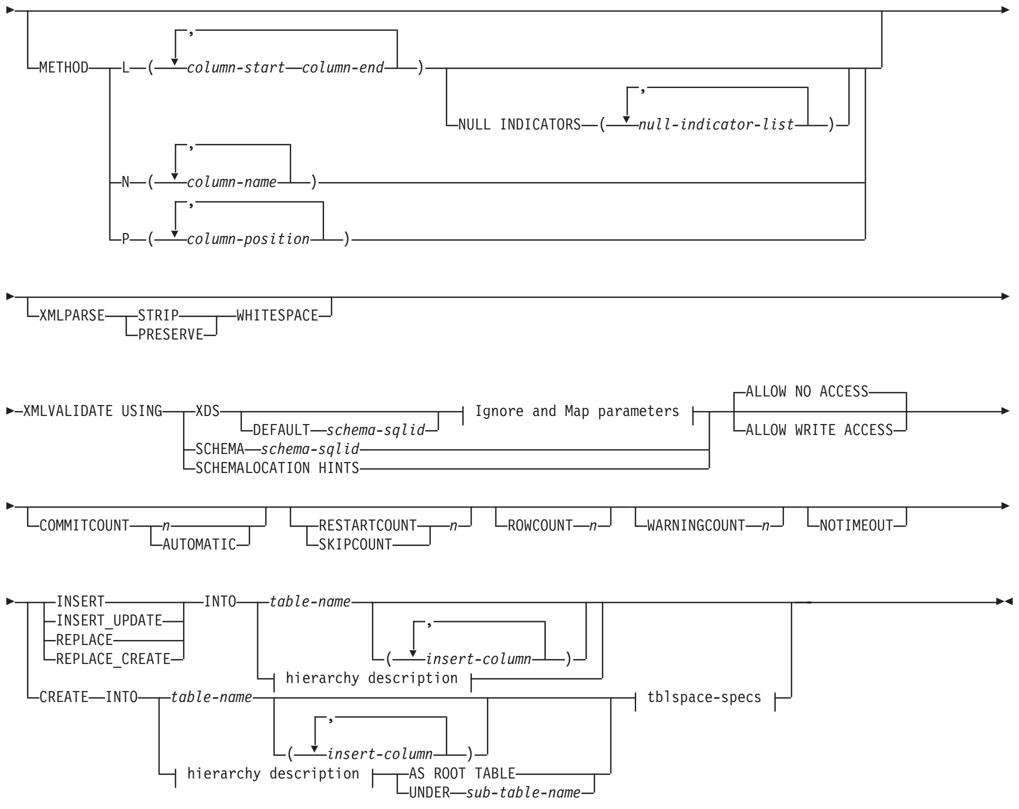
- IMPORT to a hierarchy that does not exist using the CREATE, or the REPLACE_CREATE option, requires one of the following:
  - *sysadm*
  - *dbadm*
  - CREATETAB authority on the database and USE privilege on the table space and one of:
    - IMPLICIT_SCHEMA authority on the database, if the schema name 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.

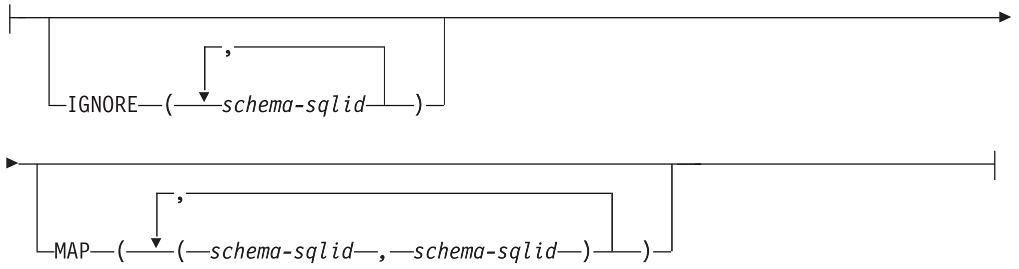
Required connection:

Command syntax:

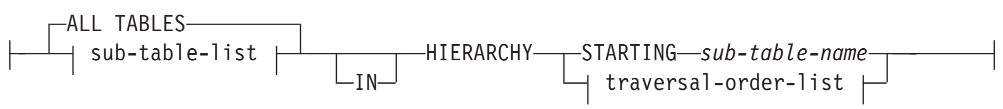
```
>>--IMPORT FROM--filename--OF--filetype--+-----------------------------+--+----------------------------+-->
                                         |            .-,--------.     |  |           .-,--------.     |
                                         |            V          |     |  |           V          |     |
                                         '-LOBS FROM----lob-path-+-----'  '-XML FROM----xml-path-+-----'

>--+-----------------------------------+------------------------------------------------------------------>
   |             .--------------.      |
   |             V              |      |
   '-MODIFIED BY----filetype-mod-+-----'
```



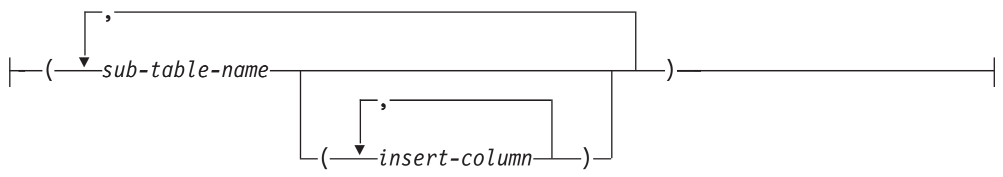
Ignore and Map parameters:



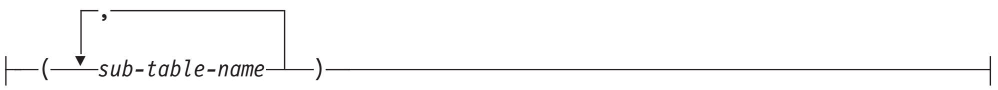
hierarchy description:



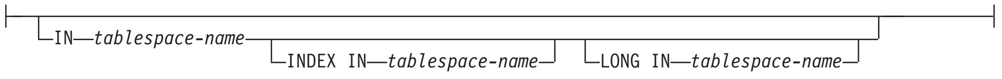
sub-table-list:



traversal-order-list:



tblspace-specs:



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:

- "Privileges, authorities, and authorization required to use import" on page 38

Related tasks:

- "Importing data" on page 38

Related reference:

- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "ADMIN_GET_MSGS table function – Retrieve messages generated by a data movement utility that is executed through the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "ADMIN_REMOVE_MSGS procedure – Clean up messages generated by a data movement utility that is executed through the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 73
- "db2look - DB2 statistics and DDL extraction tool command" in *Command Reference*
- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*

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” on page 38

Related reference:
- “IMPORT ” on page 49
- “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” on page 61

- "db2Export - Export data from a database" on page 19
- "db2Load - Load data into a table" on page 161

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++)"

File type modifiers for the import utility

Table 6. Valid file type modifiers for the import utility: All file formats

Modifier	**Description**
compound=*x*	*x* is a number between 1 and 100 inclusive. Uses nonatomic compound SQL to insert the data, and *x* statements will be attempted each time. If this modifier is specified, and the transaction log is not sufficiently large, the import operation will fail. The transaction log must be large enough to accommodate either the number of rows specified by COMMITCOUNT, or the number of rows in the data file if COMMITCOUNT is not specified. It is therefore recommended that the COMMITCOUNT option be specified to avoid transaction log overflow. This modifier is incompatible with INSERT_UPDATE mode, hierarchical tables, and the following modifiers: `usedefaults`, `identitymissing`, `identityignore`, `generatedmissing`, and `generatedignore`.
generatedignore	This modifier informs the import utility that data for all generated columns is present in the data file but should be ignored. This results in all values for the generated columns being generated by the utility. This modifier cannot be used with the `generatedmissing` modifier.
generatedmissing	If this modifier is specified, the utility assumes that the input data file contains no data for the generated columns (not even NULLs), and will therefore generate a value for each row. This modifier cannot be used with the `generatedignore` modifier.
identityignore	This modifier informs the import 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 the `identitymissing` 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 the `identityignore` modifier.

Table 6. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
lobsinfile	*lob-path* specifies the path to the files containing LOB 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. 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 IMPORT utility the list of paths to search for the LOB files while importing the data. 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/.
no_type_id	Valid only when importing into a single sub-table. Typical usage is to export data from a regular table, and then to invoke an import operation (using this modifier) to convert the data into a single sub-table.
nodefaults	If a source column for a target table column is not explicitly specified, and the table column is not nullable, default values are not loaded. Without this option, if a source column for one of the target table columns is not explicitly specified, one of the following occurs: • If a default value can be specified for a column, the default value is loaded • If the column is nullable, and a default value cannot be specified for that column, a NULL is loaded • If the column is not nullable, and a default value cannot be specified, an error is returned, and the utility stops processing.
norowwarnings	Suppresses all warnings about rejected rows.
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. IMPORT 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) 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 import fails and an error (SQLCODE SQL3525N) is returned.
seclabelname	Indicates that security labels in the input source file are indicated by their name rather than the default encoded numeric format. IMPORT 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 import fails and an error (SQLCODE SQL3525N) is returned. **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 6. Valid file type modifiers for the import utility: All file formats (continued)

Modifier	Description
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. **Note:** 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 7. Valid file type modifiers for the import 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 output data set. Converts character data from this code page to the application code page during the import operation. The following rules apply: • For pure DBCS (graphic) mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive. • `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. **Notes:** 1. The `codepage` modifier cannot be used with the `lobsinfile` modifier. 2. If data expansion occurs when the code page is converted from the application code page to the database code page, the data might be truncated and loss of data can occur.

dateformat="*x*"	*x* is the format of the date in the source file.[2] 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"`

Table 7. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
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`.

timeformat="*x*"	*x* is the format of the time in the source file.[2] 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 7. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

timestampformat="*x*"	*x* is the format of the time stamp in the source file.[2] 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 7. Valid file type modifiers for the import utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description
usegraphiccodepage	If `usegraphiccodepage` is given, the assumption is made that data being imported into graphic or double-byte character large object (DBCLOB) data fields is in the graphic code page. The rest of the data is assumed to be in the character code page. The graphic code page is associated with the character code page. IMPORT determines the character code page through either the `codepage` modifier, if it is specified, or through the code page of the application 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.
xmlchar	Specifies that XML documents are encoded in the character code page. This option is useful for processing XML documents that are encoded in the specified character code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the character code page, otherwise the row containing the document will be rejected. Note that the character codepage is the value specified by the `codepage` file type modifier, or the application codepage if it is not specified. By default, either the documents are encoded in Unicode, or they contain a declaration tag with an encoding attribute.
xmlgraphic	Specifies that XML documents are encoded in the specified graphic code page. This option is useful for processing XML documents that are encoded in a specific graphic code page but do not contain an encoding declaration. For each document, if a declaration tag exists and contains an encoding attribute, the encoding must match the graphic code page, otherwise the row containing the document will be rejected. Note that the graphic code page is the graphic component of the value specified by the `codepage` file type modifier, or the graphic component of the application code page if it is not specified. By default, documents are either encoded in Unicode, or they contain a declaration tag with an encoding attribute. **Note:** If the `xmlgraphic` modifier is specified with the IMPORT command, the XML document to be imported must be encoded in the UTF-16 code page. Otherwise, the XML document may be rejected with a parsing error, or it may be imported into the table with data corruption.

Table 8. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format

Modifier	Description
nochecklengths	If `nochecklengths` is specified, an attempt is made to import 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 imported 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.

Table 8. Valid file type modifiers for the import utility: ASC (non-delimited ASCII) file format (continued)

Modifier	Description
nullindchar=*x*	*x* is a single character. Changes the character denoting a null value to *x*. The default value of *x* is `Y`.[3] 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.
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.
	
	In the following example, `striptblanks` causes the import utility to truncate trailing blank spaces:
	`db2 import from myfile.asc of asc`
	`   modified by striptblanks`
	`   method l (1 10, 12 15) messages msgs.txt`
	`   insert into staff`
	
	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.

Table 9. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format

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.[34] If you want to explicitly specify the double quotation mark as the character string delimiter, it should be specified as follows:
	`modified by chardel""`
	
	The single quotation mark (') can also be specified as a character string delimiter. In the following example, `chardel''` causes the import utility to interpret any single quotation mark (') it encounters as a character string delimiter:
	`db2 "import from myfile.del of del`
	`   modified by chardel''`
	`   method p (1, 4) insert into staff (id, years)"`

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.[34]
	
	In the following example, `coldel;` causes the import utility to interpret any semicolon (;) it encounters as a column delimiter:
	`db2 import from myfile.del of del`
	`   modified by coldel;`
	`   messages msgs.txt insert into staff`

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.

Table 9. Valid file type modifiers for the import utility: DEL (delimited ASCII) file format (continued)

Modifier	Description

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.[34]
	
	In the following example, `decpt;` causes the import utility to interpret any semicolon (;) it encounters as a decimal point:
	`db2 "import from myfile.del of del`
	`   modified by chardel'`
	`   decpt; messages msgs.txt insert into staff"`

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 import ... 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 trailing blanks that are not inside character delimiters are removed, and a NULL is inserted into the table for all blank fields.
nochardel	The import 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 10. Valid file type modifiers for the import 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 import each row.
indexixf	Directs the utility to drop all indexes currently defined on the existing table, and to create new ones from the index definitions in the PC/IXF file. This option can only be used when the contents of a table are being replaced. It cannot be used with a view, or when a *insert-column* is specified.

Table 10. Valid file type modifiers for the import utility: IXF file format (continued)

Modifier	Description
indexschema=*schema*	Uses the specified *schema* for the index name during index creation. If *schema* is not specified (but the keyword `indexschema` *is* specified), uses the connection user ID. If the keyword is not specified, uses the schema in the IXF file.
nochecklengths	If `nochecklengths` is specified, an attempt is made to import 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 imported 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.
forcecreate	Specifies that the table should be created with possible missing or limited information after returning SQL3311N during an import operation.

Table 11. IMPORT behavior when using codepage and usegraphiccodepage

codepage=N	usegraphiccodepage	IMPORT behavior
Absent	Absent	All data in the file is assumed to be in the application code page.
Present	Absent	All data in the file is assumed to be in code page `N`. **Warning:** Graphic data will be corrupted when imported into the database if `N` is a single-byte code page.
Absent	Present	Character data in the file is assumed to be in the application code page. Graphic data is assumed to be in the code page of the application graphic data. If the application code page is single-byte, then all data is assumed to be in the application code page. **Warning:** If the application code page is single-byte, graphic data will be corrupted when imported into the database, even if the database contains graphic columns.
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 imported into the database if `N` is a single-byte code page.

Notes:

1. The import 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 import operation fails, and an error code is returned.
2. 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, \).

3. 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 ...
   ```
4. Delimiter restrictions for moving data lists restrictions that apply to the characters that can be used as delimiter overrides.
5. The following file type modifers are not allowed when importing into a nickname:
   - `indexixf`
   - `indexschema`
   - `dldelfiletype`
   - `nodefaults`
   - `usedefaults`
   - `no_type_idfiletype`
   - `generatedignore`
   - `generatedmissing`
   - `identityignore`
   - `identitymissing`
   - `lobsinfile`
6. The WSF file format is not supported for XML columns.
7. The CREATE mode is not supported for XML columns.
8. All XML data must reside in XML files that are separate from the main data file. An XML Data Specifier (XDS) (or a NULL value) must exist for each XML column in the main data file.
9. XML documents are assumed to be in Unicode format or to contain a declaration tag that includes an encoding attribute, unless the XMLCHAR or XMLGRAPHIC file type modifier is specified.
10. Rows containing documents that are not well-formed will be rejected.

11. If the XMLVALIDATE option is specified, documents that successfully validate against their matching schema will be annotated with the schema information as they are inserted. Rows containing documents that fail to validate against their matching schema will be rejected. To successfully perform the validation, the privileges held by the user invoking the import must include at least one of the following:
    - SYSADM or DBADM authority
    - USAGE privilege on the XML schema to be used in the validation

Related reference:
- "Delimiter restrictions for moving data" on page 259
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 73
- "IMPORT " on page 49

Character set and NLS considerations

Unequal code page situations, involving expansion or contraction of the character data, can sometimes occur. For example, Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) might encode different lengths for the same character. Normally, comparison of input data length to target column length is performed before reading in any data. If the input length is greater than the target length, NULLs are inserted into that column if it is nullable. Otherwise, the request is rejected. If the `nochecklengths` modifier is specified, no initial comparison is performed, and an attempt is made to import the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is imported.

Related concepts:
- "Character set and national language support" on page 206

Related reference:
- "IMPORT " on page 49

Import sessions - CLP examples

Example 1

The following example shows how to import information from `myfile.ixf` to the STAFF table:

```
   db2 import from myfile.ixf of ixf messages msg.txt insert into staff

SQL3150N  The H record in the PC/IXF file has product "DB2    01.00", date
"19970220", and time "140848".

SQL3153N  The T record in the PC/IXF file has name "myfile",
qualifier "        ", and source "            ".

SQL3109N  The utility is beginning to load data from file "myfile".

SQL3110N  The utility has completed processing.  "58" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "58".

SQL3222W  ...COMMIT of any database changes was successful.
```

```
SQL3149N  "58" rows were processed from the input file.  "58" rows were
successfully inserted into the table.  "0" rows were rejected.
```

Example 3 (Importing into a Table with an Identity Column)

TABLE1 has 4 columns:
- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.

```
db2 import from datafile1.del of del replace into table1
```

To import DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:

```
db2 import from datafile1.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
db2 import from datafile1.del of del modified by identityignore
   replace into table1
```

To import DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

```
db2 import from datafile2.del of del replace into table1 (c1, c3, c4)
db2 import from datafile2.del of del modified by identitymissing
   replace into table1
```

If DATAFILE1 is imported into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be inserted, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 4 (Importing Using Null Indicators)

TABLE1 has 5 columns:
- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4

- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:
- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE5 positions 23 to 23
- ELE3 positions 24 to 27
- ELE4 positions 28 to 31
- ELE6 positions 32 to 32
- ELE6 positions 33 to 40

Data Records:
```
1...5....10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    wxyzN
Test data 4,5 and 6 WWN6789    Y
```

The following command imports records from ASCFILE1 into TABLE1:

```
db2 import from ascfile1 of asc
method L (1 20, 21 22, 24 27, 28 31)
null indicators (0, 0, 23, 32)
insert into table1 (col1, col5, col2, col3)
```

Notes:
1. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
2. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(........)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
3. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
4. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Related concepts:
- "Import Overview" on page 35
- "Importing large objects (LOBS)" on page 46
- "Importing user-defined distinct types (UDTs)" on page 47
- "Importing XML data" on page 40

Related tasks:
- "Importing data" on page 38

Related reference:
- Appendix B, "Differences between the import and load utility," on page 281

File type modifiers for the import utility

Chapter 3. Load

This chapter describes the DB2 load utility, which moves data from files, named pipes, devices or a cursor into a DB2 table. These data sources can reside either on the database partition where the database resides, or on a remotely connected client. If the table receiving the new data already contains data, you can replace or append to the existing data.

The following topics are covered:

Load overview

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data. The utility can handle most data types, including large objects (LOBs) and user-defined types (UDTs). The load utility is faster than the import utility, because it writes formatted pages directly into the database, while the import utility performs SQL INSERTs. The load utility does not fire triggers, and does not perform referential or table constraints checking (other than validating the uniqueness of the indexes).

The load process consists of four distinct phases (see Figure 1):

- Load, during which the data is written to the table.

  During the load phase, data is loaded into the table, and index keys and table statistics are collected, if necessary. Save points, or points of consistency, are established at intervals specified through the SAVECOUNT parameter in the LOAD command. Messages are generated, indicating how many input rows were successfully loaded at the time of the save point. If a failure occurs, you can restart the load operation; the RESTART option automatically restarts the load operation from the last successful consistency point. The TERMINATE option rolls back the failed load operation.

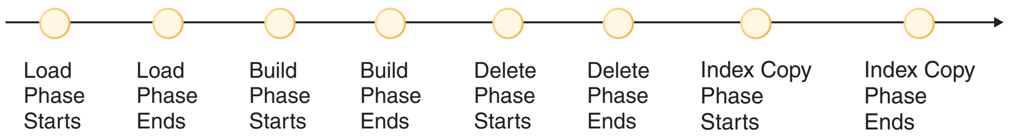


Figure 1. The Four Phases of the Load Process: Load, Build, Delete, and Index Copy. While the load operation is taking place, the target table is in the load in progress state. If the table has constraints, the table will also be in the set integrity pending state. If the ALLOW READ ACCESS option was specified, the table will also be in the read access only state.

- Build, during which indexes are produced.

  During the build phase, indexes are produced based on the index keys collected during the load phase. The index keys are sorted during the load phase, and index statistics are collected (If STATISTICS USE PROFILE option was specified, and profile indicates collecting index stats). The statistics are similar to those collected through the RUNSTATS command. If a failure occurs during the build phase, the RESTART option automatically restarts the load operation at the appropriate point.
- Delete, during which the rows that caused a unique key violation are removed from the table. Unique key violations are placed into the exception table, if one was specified, and messages about rejected rows are written to the message file. Following the completion of the load process, review these messages, resolve any problems, and insert corrected rows into the table.

  Do not attempt to delete or to modify any temporary files created by the load utility. Some temporary files are critical to the delete phase. If a failure occurs during the delete phase, the RESTART option automatically restarts the load operation at the appropriate point.

  Note: Each deletion event is logged. If you have a large number of records that violate the uniqueness condition, the log could fill up during the delete phase.
- Index copy, during which the index data is copied from a system temporary table space to the original table space. This will only occur if a system

temporary table space was specified for index creation during a load operation with the READ ACCESS option specified.

Note: After you invoke the load utility, you can use the LIST UTILITIES command to monitor the progress of the load operation. For more information, refer to LIST UTILITIES command.

The following information is required when loading data:
- The path and the name of the input file, named pipe, or device.
- The name or alias of the target table.
- The format of the input source. This format can be DEL, ASC, PC/IXF, or CURSOR.
- Whether the input data is to be appended to the table, or is to replace the existing data in the table.
- A message file name, if the utility is invoked through the application programming interface (API), **db2Load**.

You can also specify:
- That the data to be loaded resides on the client, if the load utility is invoked from a remotely connected client.
- The method to use for loading the data: column location, column name, or relative column position.
- How often the utility is to establish consistency points. Use the SAVECOUNT parameter to specify this value. If this parameter is specified, a load restart operation will start at the last consistency point, instead of at the beginning.
- The names of the table columns into which the data is to be inserted.
- Whether or not pre-existing data in the table can be queried while the load operation is in progress.

  Note: This can be accomplished by using the READ ACCESS option and is not supported when the load utility is invoked in REPLACE mode.
- Whether the load operation should wait for other utilities or applications to finish using the table or force the other applications off before proceeding.
- An alternate system temporary table space in which to build the index.

  Note: This is only supported when the READ ACCESS option is specified with a full index rebuild.
- The paths and the names of the input files in which LOBs are stored. The `lobsinfile` modifier tells the load utility that all LOB data is being loaded from files.
- A message file name. During operations such as exporting, importing, loading, binding, or restoring data, you can specify that message files be created to contain the error, warning, and informational messages associated with those operations. Specify the name of these files with the MESSAGES parameter. These message files are standard ASCII text files. To print them, use the printing procedure for your operating system; to view them, use any ASCII editor.

  Notes:
  1. You can only view the contents of a message file after the operation is finished.
  2. Each message in a message file begins on a new line and contains information provided by the DB2 message retrieval facility.

- Whether column values being loaded have implied decimal points. The `implieddecimal` modifier tells the load utility that decimal points are to be applied to the data as it enters the table. For example, the value 12345 is loaded into a DECIMAL(8,2) column as 123.45, not 12345.00.
- Whether the utility should modify the amount of free space available after a table is loaded. Additional free space permits INSERT and UPDATE growth to the table following the completion of a load operation. Reduced free space keeps related rows more closely together and can enhance table performance.
- Whether statistics are to be gathered during the load process. This option is only supported if the load operation is running in REPLACE mode.

  If data is appended to a table, statistics are not collected. To collect current statistics on an appended table, invoke the runstats utility following completion of the load process. If gathering statistics on a table with a unique index, and duplicate keys are deleted during the delete phase, statistics are not updated to account for the deleted records. If you expect to have a significant number of duplicate records, do not collect statistics during the load operation. Instead, invoke the runstats utility following completion of the load process.
- Whether to collect statistics during the load operation. Statistics are collected according to the profile defined for the table. The profile must be created by the RUNSTATS command before the LOAD command is executed. If the profile does not exist and the load operation is instructed to collect statistics according to the profile, the load will fail, and an error message will be returned.
- Whether to keep a copy of the changes made. This is done to enable rollforward recovery of the database. This option is not supported if rollforward recovery is disabled for the database; that is, if the database configuration parameters *logarchmeth1* and *logarchmeth2* are set to OFF. If no copy is made, and rollforward recovery is enabled, the table space is left in backup pending state at the completion of the load operation.

  Logging is required for fully recoverable databases. The load utility almost completely eliminates the logging associated with the loading of data. In place of logging, you have the option of making a copy of the loaded portion of the table. If you have a database environment that allows for database recovery following a failure, you can do one of the following:
  - Explicitly request that a copy of the loaded portion of the table be made.
  - Take a backup of the table spaces in which the table resides immediately after the completion of the load operation.

  If you are loading a table that already contains data, and the database is non-recoverable, ensure that you have a backed-up copy of the database, or the table spaces for the table being loaded, before invoking the load utility, so that you can recover from errors.

  If you want to perform a sequence of multiple load operations on a recoverable database, the sequence of operations will be faster if you specify that each load operation is non-recoverable, and take a backup at the end of the load sequence, than if you invoke each of the load operations with the COPY YES option. You can use the NONRECOVERABLE option to specify that a load transaction is to be marked as non-recoverable, and that it will not be possible to recover it by a subsequent rollforward operation. 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 rollforward operation is completed, such a table can only be dropped (see Figure 2 on page 105). 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.

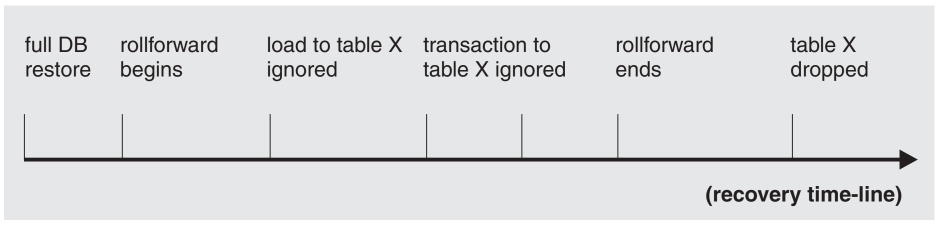


Figure 2. Non-recoverable Processing During a Roll Forward Operation

- Whether to log all index modifications. If the database configuration parameter *logindexbuild* is set, and if the load operation is invoked with the COPY YES recoverability option and the INCREMENTAL indexing option, the load will log all index modifications. The benefit of using these options is that when you roll forward through the log records for this load, you will also recover the indexes (whereas normally the indexes would not be recovered unless the load had used REBUILD indexing mode).
- The fully qualified path to be used when creating temporary files during a load operation. The name is specified by the TEMPFILES PATH parameter of the LOAD command. The default value is the database path. The path resides on the server machine, and is accessed by the DB2 instance exclusively. Therefore, any path name qualification given to this parameter must reflect the directory structure of the server, not the client, and the DB2 instance owner must have read and write permission on the path. This is true even if you are the instance owner. If you are not the instance owner, you must specify a location that is writable by the instance owner.

Changes to Previous Load Behavior Introduced in DB2 V9.1

Following is a summary of changes to load behavior introduced in DB2 Version 9.1:

- In DB2 UDB Version 8, if a lob file is not found, the row is rejected if the column is not nullable, or NULL if the column is nullable. In DB2 Version 9.1, if a lob file is not found, the row is rejected regardless of the nullability of the column.
- In DB2 Version 9.1, message SQL3040N is split into two distinct messages. SQL3040N is returned for lobfile errors and SQL3235N is returned for lob path errors. The invalid file name or path name is indicated in the message.
- In DB2 UDB Version 8, if the LOB Location Specifier (LLS) contains a path, for example, the LLS is /home/try/newlob.001.12.345/ and the path is invalid, SQL3040N reason code 6 is returned and the utility exits immediately. In DB2 Version 9.1, the row is rejected and processing continues. In DB2 V9.1, the exported LLS never contains a path name.
- In DB2 UDB Version 8, if LOBSINFILE is specified, and LOBS FROM is specified, the specified lob directory is searched. If LOBS FROM is not specified, the current working directory is searched. In DB2 V9.1, if LOBSINFILE is specified and
  - loading from a local client and LOBS FROM is specified, the specified lob directory is searched first, then the current working directory is searched.
  - loading from a local client and LOBS FROM is not specified, the path where the input data files reside is searched first, then the current working directory is searched. If there are multiple input data files, the utility must verify

whether the input data files come from the same path. If not, an SQL3052N message is returned and you are asked to specify the LOBS FROM option.
  - loading from a remote client and LOBS FROM is specified, the specified lob directory is searched.
  - loading from a remote client and LOBS FROM is not specified, the data file directory is searched.
  - remote load (load with CLIENT option is specified) and LOBS FROM is specified, the specified lob directory is searched.
  - remote load and LOBS FROM is not specified, an SQL3052N message is returned and you are asked to specify the LOBS FROM option.
- In DB2 UDB Version 8, a LOAD FROM CURSOR operation does not allow loading of mismatched codepage tables. An example of a mismatch includes a non-Unicode database with Unicode tables. In DB2 Version 9.1, codepage conversion is supported in the LOAD FROM CURSOR operation.
- In DB2 UDB Version 8 the FILE_TRANSFER_CMD handles the fetching of data. In DB2 Version 9.1 the fetching of data from the source database is handled by the load utility using a user exit process.

Changes to previous load behavior introduced in DB2 UDB Version 8

Following is a summary of syntax changes and changes to load behavior introduced in DB2 UDB Version 8:

- Prior to DB2 UDB Version 8, load required exclusive access to table spaces that contained objects belonging to the table being loaded. In DB2 UDB Version 8, load operates at the table level and no longer requires exclusive access to the table space. Load will place a lock only on the table objects associated with the load operation taking place. Concurrent access to other table objects in the same table spaces is permitted.

  Note: Prior to DB2 UDB Version 8, when the COPY NO option was specified on a recoverable database, the table space was put in backup pending state only after the load operation was committed. In DB2 UDB Version 8, the table space will be placed in backup pending state when the load operation begins and will remain in that state even if the load operation fails and is rolled back. As in previous releases, when the COPY NO option is specified and load operation completes successfully, the rollforward utility will put dependent table spaces in restore pending state during a rollforward operation.
- You can also specify that users have read access to the data that existed in the table prior to the load. This means that after the load operation has completed, you will not be able to view the new data if there are constraints on the table and integrity checking has not been completed. You can also specify that the index be rebuilt in a separate table space during a load operation by specifying the READ ACCESS and INDEXING MODE REBUILD options. The index will be copied back to the original table space during the index copy phase which occurs after the other phases of the load operation.
- The functionality of the LOAD QUERY command has been expanded and it now returns the table state of the target into which data is being loaded in addition to the status information it previously included on a load operation in progress. The LOAD QUERY command might also be used to query the table state whether or not a load operation is in progress on that table.
- Extent allocations in DMS table spaces are now logged. The LOAD command will now write two log records for every extent it allocates in a DMS table space.

Also, when the READ ACCESS and INDEXING MODE INCREMENTAL options are specified, some log records will be written while data is being incrementally inserted into the index.

- Dependent table spaces will no longer be quiesced prior to a load operation. When the COPY NO option is specified, the new table space state *load in progress* will be used. The load in progress table space state prevents the backup of dependent tables during a load operation. The load in progress table space state is different from the load in progress table state in that all load operations use the load in progress table state, but load operations with the COPY NO option specified also use the load in progress table space state.
- When executing a load operation with the ALLOW READ ACCESS and INDEXING MODE REBUILD options, a new copy of the indexes is created in addition to the original indexes. This means that the space requirement for the index table space might have to be doubled. To avoid this, the USE TABLESPACE option can be used to specify a temporary table space for the storage of new indexes. After the new indexes are built in the temporary table space, the target table is taken offline before the new indexes are copied into the target table space.
- Calls to quiesce table spaces from the LOAD command have been removed. If you quiesce table spaces in exclusive mode prior to a load operation, you will now have to explicitly remove the table spaces from the quiesced exclusive state. In previous releases, after issuing the following commands LOAD would have implicitly reset the quiesced table spaces and made them accessible to other applications:

  ```
  quiesce tablespaces for table t1 exclusive
  load from data.del of del insert into t1
  ```

  In DB2 UDB Version 8, you must issue the following command to remove the table space from the quiesced exclusive state:

  ```
  quiesce tablespaces for table t1 reset
  ```
- A LOCK WITH FORCE option has been added to the LOAD command. It allows you to force other applications to release locks they have on a table and to allow the load operation to proceed and acquire the locks it needs.
- The load utility now has the ability to load from an SQL statement, using the CURSOR file type.
- Loading data that resides on a remotely connected client is now supported under the following conditions:
  - The database that the client is connected to is in a partitioned database environment.
  - The database that the client is connected to is cataloged against an already cataloged database.
- Loading data into multidimensional clustering (MDC) tables is supported.
- Prior to DB2 UDB Version 8, following a load operation the target table remained in set integrity pending state if it contained generated columns. The load utility will now generate column values, and you are no longer required issue the SET INTEGRITY statement after a load operation.
- Tables can be loaded into a multi-partition database using the load API (**db2Load**).

Related concepts:

- "Rollforward recovery" in *Data Recovery and High Availability Guide and Reference*

- "Load considerations for partitioned tables" on page 126
- "Loading data in a partitioned database environment - hints and tips" on page 237

Related tasks:

- "Loading data" on page 110
- "Loading data in a partitioned database environment" on page 219

Related reference:

- "LIST UTILITIES command" in *Command Reference*
- "Load configuration options for partitioned database environments" on page 229
- "RUNSTATS command" in *Command Reference*

Parallelism and loading

The load utility takes advantage of a hardware configuration in which multiple processors or multiple storage devices are used, such as in a symmetric multiprocessor (SMP) environment. There are several ways in which parallel processing of large amounts of data can take place using the load utility. One way is through the use of multiple storage devices, which allows for I/O parallelism during the load operation (see Figure 3). Another way involves the use of multiple processors in an SMP environment, which allows for intra-partition parallelism (see Figure 4). Both can be used together to provide even faster loading of data.

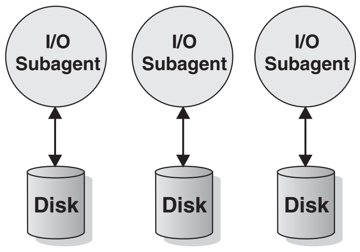


Figure 3. Taking Advantage of I/O Parallelism When Loading Data

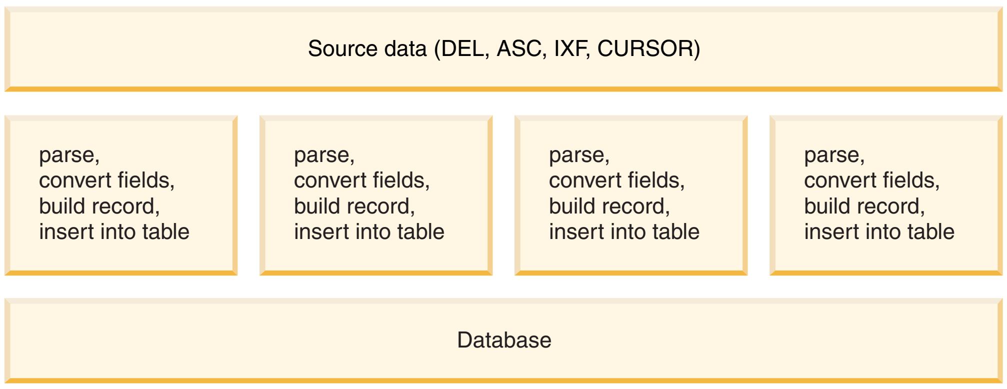

Source data (DEL, ASC, IXF, CURSOR)
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
parse, convert fields, build record, insert into table
Database

Figure 4. Taking Advantage of Intra-partition Parallelism When Loading Data

Related concepts:

- "Load considerations for partitioned tables" on page 126
- "Load overview" on page 102

- “Optimizing load performance” on page 207

Related tasks:

- “Loading data in a partitioned database environment” on page 219

Related reference:

- “Load configuration options for partitioned database environments” on page 229

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 161
- “LOAD ” on page 132

Loading data

The load utility is capable of efficiently moving large quantities of data into newly created tables, or into tables that already contain data.

Prerequisites:

Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be loaded. Since the utility will issue a COMMIT statement, you should complete all transactions and release all locks by issuing either a COMMIT or a ROLLBACK statement before invoking the load utility.

Data is loaded in the sequence that appears in the input file, except when using multi-dimensional clustering (MDC) tables, partitioned tables, or the ANYORDER clause. If a particular sequence is desired, sort the data before attempting a load operation.

If clustering is required, the data should be sorted on the clustering index prior to loading. When loading data into multidimensional clustered tables (MDC), sorting is not required prior to the load operation, and data is clustered according to the MDC table definition.

When loading data into partitioned tables, sorting is not required prior to the load operation, and data is partitioned according to the table definition.

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.
- 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.
- If the REPLACE option is specified, the 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.

Restrictions:

The following restrictions apply to the load utility:
- Loading data into nicknames is not supported.
- Loading data into typed tables, or tables with structured type columns, is not supported.
- Loading data into declared temporary tables is not supported.
- You cannot create or drop tables in a table space that is in backup pending state.
- You cannot load data into a database accessed through DB2 Connect or a server level prior to DB2 Version 2. Options that are only available with the current cannot be used with a server from the previous release.
- If an error occurs during a LOAD REPLACE operation, the original data in the table is lost. Retain a copy of the input data to allow the load operation to be restarted.
- Triggers are not activated on newly loaded rows. Business rules associated with triggers are not enforced by the load utility.
- Loading data into tables containing XML columns is not supported.

The following restrictions apply to the load utility when loading into a partitioned table:
- Consistency points are not supported.
- Loading data into a subset of data partitions while keeping the remaining data partitions fully online is not supported.
- The exception table used by a load operation or a set integrity pending operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.

Procedure:

The load utility can be invoked through the command line processor (CLP), the Load wizard in the Control Centre, or an application programming interface (API), **db2Load**.

The following is an example of the LOAD command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   insert into userid.staff copy yes use tsm data buffer 4000
```

In this example:
- Any warning or error messages are placed in the `staff.msgs` file.
- A copy of the changes made is stored in Tivoli® Storage Manager (TSM).
- Four thousand pages of buffer space are to be used during the load operation.

The following is another example of the LOAD command issued through the CLP:

```
db2 load from stafftab.ixf of ixf messages staff.msgs
   tempfiles path /u/myuser replace into staff
```

In this example:

- The table data is being replaced.
- The TEMPFILES PATH parameter is used to specify `/u/myuser` as the server path into which temporary files will be written.

Note: These examples use relative path names for the load input file. Relative path names are only allowed on calls from a client on the same database partition as the database. The use of fully qualified path names is recommended.

To open the Load wizard:

1. From the Control Center, expand the object tree until you find the Tables folder.
2. Click on the Tables folder. Any existing tables are displayed in the pane on the right side of the window (the contents pane).
3. Right-click on the table you want in the contents pane, and select Load from the pop-up menu. The Load wizard opens.
4. Specify the required information on each page of the wizard to successfully load your data.

Detailed information about the Load wizard is provided through its online help facility.

After you invoke the load utility, you can use the LIST UTILITIES command to monitor the progress of the load operation. In the case of a load operation performed in either INSERT mode, REPLACE mode, or RESTART mode, detailed progress monitoring support is available. Issue the LIST UTILITIES command with the SHOW DETAILS option to view detailed information about the current load phase. Details are not available for a load operation performed in TERMINATE mode. The LIST UTILITIES command will simply show that a load terminate utility is currently running.

A load operation maintains unique constraints, range constraints for partitioned tables, generated columns, and LBAC security rules. For all other constraints the table is placed in the SET INTEGRITY PENDING state at the beginning of a load operation. After the load operation is complete, the SET INTEGRITY statement must be used to take the table out of SET INTEGRITY PENDING state.

Related concepts:

- "Load considerations for partitioned tables" on page 126
- "Load overview" on page 102
- "Loading data in a partitioned database environment - hints and tips" on page 237

Related tasks:

- "Troubleshooting load issues" in *Troubleshooting Guide*
- "Loading data in a partitioned database environment" on page 219

Related reference:

- "Tivoli Storage Manager" in *Data Recovery and High Availability Guide and Reference*
- "Load configuration options for partitioned database environments" on page 229
- "LIST UTILITIES command" in *Command Reference*
- "LOAD " on page 132

Read access load operations

The load utility provides two options that control the amount of access other applications have to a table being loaded. The ALLOW NO ACCESS option locks the table exclusively and allows no access to the table data while the table is being loaded. This is the default behavior. The ALLOW READ ACCESS option prevents all write access to the table by other applications, but allows read access to pre-loaded data. This section deals with the ALLOW READ ACCESS option.

Table data and index data that exist prior to the start of a load operation are visible to queries while the load operation is in progress. Consider the following example:

1. Create a table with one integer column:
   ```
   create table ED (ed int)
   ```
2. Load three rows:
   ```
   load from File1 of del insert into ED
   ...
   Number of rows read         = 3
   Number of rows skipped      = 0
   Number of rows loaded       = 3
   Number of rows rejected     = 0
   Number of rows deleted      = 0
   Number of rows committed    = 3
   ```
3. Query the table:
   ```
   select * from ED

   ED
   -----------
             1
             2
             3

     3 record(s) selected.
   ```
4. Perform a load operation with the ALLOW READ ACCESS option specified and load two more rows of data:
   ```
   load from File2 of del insert into ED allow read access
   ```
5. At the same time, on another connection query the table while the load operation is in progress:
   ```
   select * from ED

   ED
   -----------
             1
             2
             3

     3 record(s) selected.
   ```
6. Wait for the load operation to finish and then query the table:
   ```
   select * from ED

   ED
   -----------
             1
             2
             3
             4
             5

     5 record(s) selected.
   ```

The ALLOW READ ACCESS option is very useful when loading large amounts of data because it gives users access to table data at all times, even when the load operation is in progress or after a load operation has failed. The behavior of a load operation in ALLOW READ ACCESS mode is independent of the isolation level of the application. That is, readers with any isolation level can always read the pre-existing data, but they will not be able to read the newly loaded data until the load operation has finished.

Read access is provided throughout the load operation except for two instances at the beginning and end of the operation.

Firstly, the load operation acquires a special Z-lock for a short duration of time near the end of its setup phase. If an application holds an incompatible lock on the table prior to the load operation requesting this special Z-lock, then the load operation waits a finite amount of time for this incompatible lock to be released before timing out and failing. The amount of time is determined by the LOCKTIMEOUT database configuration parameter. If the LOCK WITH FORCE option is specified then the load operation forces other applications off to avoid timing out. The load operation acquires the special Z-lock, commits the phase, releases the lock and then continues onto the load phase. Any application that requests a lock on the table for reading after the start of the load operation in ALLOW READ ACCESS mode, is granted the lock and it does not conflict with this special Z-lock. New applications attempting to read existing data from the target table are able to do so.

Secondly, before data is committed at the end of the load operation, the utility acquires an exclusive lock (Z-lock) on the table. The load utility waits until all applications that hold locks on the table, release them. This can cause a delay before the data is committed. The LOCK WITH FORCE option is used to force off conflicting applications, and allow the load operation to proceed without having to wait. Usually, a load operation in ALLOW READ ACCESS mode acquires an exclusive lock for a short amount of time; however, if the USE <tablespaceName> option is specified, the exclusive lock lasts for the entire period of the index copy phase.

Notes:

1. If a load operation is aborted, it remains at the same access level that was specified when the load operation was issued. So, if a load operation in ALLOW NO ACCESS mode aborts, the table data is inaccessible until a load terminate or a load restart is issued. If a load operation in ALLOW READ ACCESS mode aborts, the pre-loaded table data is still accessible for read access.
2. If the ALLOW READ ACCESS option was specified for an aborted load operation, it can also be specified for the load restart or load terminate operation. However, if the aborted load operation specified the ALLOW NO ACCESS option, the ALLOW READ ACCESS option cannot be specified for the load restart or load terminate operation.

The ALLOW READ ACCESS option is not supported if:

- The REPLACE option is specified. Since a load replace operation truncates the existing table data before loading the new data, there is no pre-existing data to query until after the load operation is complete.
- The indexes have been marked invalid and are waiting to be rebuilt. Indexes can be marked invalid in some rollforward scenarios or through the use of the **db2dart** command.

- The INDEXING MODE DEFERRED option is specified. This mode marks the indexes as requiring a rebuild.
- An ALLOW NO ACCESS load operation is being restarted or terminated. Until it is brought fully online, a load operation in ALLOW READ ACCESS mode cannot take place on the table.
- A load operation is taking place to a table that is in SET INTEGRITY PENDING NO ACCESS state. This is also the case for multiple load operations on tables with constraints. A table is not brought online until the SET INTEGRITY statement is issued.

Generally, if table data is taken offline, read access is not available during a load operation until the table data is back online.

Related concepts:
- "Building indexes" on page 115
- "Checking for integrity violations following a load operation" on page 121
- "Table locking, table states and table space states" on page 203

Building indexes

Indexes are built during the build phase of a load operation. There are four indexing modes that can be specified in the LOAD command:

1. REBUILD. All indexes are rebuilt.
2. INCREMENTAL. Indexes are extended with new data.
3. AUTOSELECT. The load utility automatically decides between REBUILD or INCREMENTAL mode. This is the default.

   Note: You might decide to explicitly choose an indexing mode because the behavior of the REBUILD and INCREMENTAL modes are quite different.
4. DEFERRED. The load utility does not attempt index creation if this mode is specified. Indexes are marked as needing a refresh, and a rebuild might be forced the first time they are accessed. This option is not compatible with the ALLOW READ ACCESS option because it does not maintain the indexes and index scanners require a valid index.

Load operations that specify the ALLOW READ ACCESS option require special consideration in terms of space usage and logging depending on the type of indexing mode chosen. When the ALLOW READ ACCESS option is specified, the load utility keeps indexes available for queries even while they are being rebuilt.

When a load operation in ALLOW READ ACCESS mode specifies the INDEXING MODE INCREMENTAL option, the load utility writes some log records that protect the integrity of the index tree. The number of log records written is a fraction of the number of inserted keys and is a number considerably less than would be needed by a similar SQL insert operation. A load operation in ALLOW NO ACCESS mode with the INDEXING MODE INCREMENTAL option specified writes only a small log record beyond the normal space allocation logs.

When a load operation in ALLOW READ ACCESS mode specifies the INDEXING MODE REBUILD option, new indexes are built as a *shadow* either in the same table space as the original index or in a system temporary table space. The original indexes remain intact and are available during the load operation and are only replaced by the new indexes at the end of the load operation while the table is exclusively locked. If the load operation fails and the transaction is rolled back, the original indexes remain intact.

Building new indexes in the same table space as the original

By default, the *shadow* index is built in the same table space as the original index. Since both the original index and the new index are maintained simultaneously, there must be sufficient table space to hold both indexes at the same time. If the load operation is aborted, the extra space used to build the new index is released. If the load operation commits, the space used for the original index is released and the new index becomes the current index. When the new indexes are built in the same table space as the original indexes, replacing the original indexes takes place almost instantaneously.

If the indexes are built in a DMS table space, you cannot see the new *shadow* index. If the indexes are built within an SMS table space, you can see index files in the table space directory with the `.IN1` suffix and the `.INX` suffix. These suffixes do not indicate which is the original index and which is the *shadow* index.

Building new indexes in a system temporary table space

The new index can be built in a system temporary table space to avoid running out of space in the original table space. The USE <tablespaceName> option allows the indexes to be rebuilt in a system temporary table space when using INDEXING MODE REBUILD and ALLOW READ ACCESS options. The system temporary table can be an SMS or a DMS table space, but the page size of the system temporary table space must match the page size of the original index table space.

The USE <tablespaceName> option is ignored if the load operation is not in ALLOW READ ACCESS mode, or if the indexing mode is incompatible. The USE <tablespaceName> option is only supported for the INDEXING MODE REBUILD or INDEXING MODE AUTOSELECT options. If the INDEXING MODE AUTOSELECT option is specified and the load utility selects incremental maintenance of the indexes, the USE <tablespaceName> option is ignored.

A load restart operation can use an alternate table space for building an index even if the original load operation did not use an alternate table space. A load restart operation cannot be issued in ALLOW READ ACCESS mode if the original load operation was not issued in ALLOW READ ACCESS mode. Load terminate operations do not rebuild indexes, so the USE <tablespaceName> option is ignored.

During the build phase of the load operation, the indexes are built in the system temporary table space. Then, during the index copy phase, the index is copied from the system temporary table space to the original index table space. To make sure that there is sufficient space in the original index table space for the new index, space is allocated in the original table space during the build phase. So, if the load operation runs out of index space, it will do so during the build phase. If this happens, the original index is not lost.

The index copy phase occurs after the build and delete phases. Before the index copy phase begins, the table is locked exclusively. That is, it is unavailable for read access throughout the index copy phase. Since the index copy phase is a physical copy, the table might be unavailable for a significant amount of time.

Note: If either the system temporary table space or the index table space are DMS table spaces, the read from the system temporary table space can cause random I/O on the system temporary table space and can cause a delay. The write to the index table space is still optimized and the DISK_PARALLELISM values are used.

Related concepts:

- "Load overview" on page 102
- "Read access load operations" on page 113

Using load with identity columns

The load utility can be used to load data into a table containing an identity column. If no identity-related file type modifiers are used, the utility works according to the following rules:

- If the identity column is GENERATED ALWAYS, an identity value is generated for a table row whenever the corresponding row in the input file is missing a value for the identity column, or a NULL value is explicitly given. If a non-NULL value is specified for the identity column, the row is rejected (SQL3550W).
- If the identity column is GENERATED BY DEFAULT, the load utility makes use of user-supplied values, if they are provided; if the data is missing or explicitly NULL, a value is generated.

The load utility does not perform any extra validation of user-supplied identity values beyond what is normally done for values of the identity column's data type (that is, SMALLINT, INT, BIGINT, or DECIMAL). Duplicate values are not reported.

The assignment of identity column values is managed in parallel by the load utility. Because of this the load utility cannot guarantee that identity column values are assigned to rows in the same order that these rows appear in the datafile. Identity column values are therefore assigned in arbitrary order. The exception to this rule occurs when the CPU_PARALLELISM 1 option is specified in a single-partition database. In this case, rows are not processed in parallel, resulting in identity column values being implicitly assigned in the same order that rows appear in the datafile parameter.

For a multi-partition database, when an identity column is in the distribution key for a table, or the identity column is referenced in a generated column that is part of the distribution key, and the `identityoverride` modifier is not specified, every loading database partition must be in the load phase in order for a RESTART operation to occur. All of the database partitions must be in the load phase because hashing of rows during the restarted load might be different from the hashing in the initial load, due to the dependence on the identity column. In this case, you usually need to use the TERMINATE option to terminate the load operation.

Three (mutually exclusive) file type modifiers are supported by the load utility to simplify its use with tables that contain an identity column:

- The `identitymissing` modifier makes loading a table with an identity column more convenient if the input data file does not contain any values (not even NULLS) for the identity column. For example, consider a table defined with the following SQL statement:

```
create table table1 (c1 varchar(30),
                     c2 int generated by default as identity,
                     c3 decimal(7,2),
                     c4 char(1))
```

If you want to load TABLE1 with data from a file (`load.del`) that has been exported from a table that does not have an identity column, see the following example:

```
Robert, 45.2, J
Mike, 76.9, K
Leo, 23.4, I
```

One way to load this file would be to explicitly list the columns to be loaded through the LOAD command as follows:

```
db2 load from load.del of del replace into table1 (c1, c3, c4)
```

For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of loading the file is to use the `identitymissing` file type modifier as follows:

```
db2 load from load.del of del modified by identitymissing
   replace into table1
```

- The `identityignore` modifier is in some ways the opposite of the `identitymissing` modifier: it indicates to the load utility that even though the input data file contains data for the identity column, the data should be ignored, and an identity value should be generated for each row. For example, a user might want to load TABLE1, as defined above, from a data file (`load.del`) containing the following data:

```
Robert, 1, 45.2, J
Mike, 2, 76.9, K
Leo, 3, 23.4, I
```

If the user-supplied values of `1`, `2`, and `3` are not used for the identity column, you can issue the following LOAD command:

```
db2 load from load.del of del method P(1, 3, 4)
   replace into table1 (c1, c3, c4)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The `identityignore` modifier simplifies the syntax as follows:

```
db2 load from load.del of del modified by identityignore
   replace into table1
```

- The `identityoverride` modifier is used for loading user-supplied values into a table with a GENERATED ALWAYS identity column. This can be quite useful when migrating data from another database system, and 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 are rejected (SQL3116W).

Note: When using this modifier, it is possible to violate the uniqueness property of GENERATED ALWAYS columns.

Related concepts:

- "Identity columns" in *Administration Guide: Planning*

Using load with generated columns

The load utility can be used to load data into a table containing (non-identity) generated columns. The column values are generated by this utility.

Note: If you initiate a load operation between a Version 7 or earlier client and a Version 8 or later server, the load utility will place tables with generated columns in the set integrity pending state.

If a table has been placed in set integrity pending state because a Version 7 or earlier client was used to load data into a table with generated columns, the following statement will take the table out of set integrity pending state and force the generation of values:

```
SET INTEGRITY FOR tablename IMMEDIATE CHECKED FORCE GENERATED;
```

If no generated column-related file type modifiers are used, the load utility works according to the following rules:

- Values are created for generated columns when the corresponding row of the data file is missing a value for the column or a NULL value is supplied. If a non-NULL value is supplied for a generated column, the row is rejected (SQL3550W).
- If a NULL value is created for a generated column that is not nullable, the entire row of data is rejected (SQL0407N). This could occur if, for example, a non-nullable generated column is defined as the sum of two table columns that include NULL values in the data file.

Three (mutually exclusive) file type modifiers are supported by the load utility to simplify its use with tables that contain generated columns:

- The `generatedmissing` modifier makes loading a table with generated columns more convenient if the input data file does not contain any values (not even NULLS) for all generated columns present in the table. For example, consider a table defined with the following SQL statement:

  ```
  CREATE TABLE table1 (c1 INT,
                       c2 INT,
                       g1 INT GENERATED ALWAYS AS (c1 + c2),
                       g2 INT GENERATED ALWAYS AS (2 * c1),
                       c3 CHAR(1))
  ```

  If you want to load TABLE1 with data from a file (`load.del`) that has been exported from a table that does not have any generated columns, see the following example:

  ```
  1, 5, J
  2, 6, K
  3, 7, I
  ```

  One way to load this file would be to explicitly list the columns to be loaded through the LOAD command as follows:

  ```
  DB2 LOAD FROM load.del of del REPLACE INTO table1 (c1, c2, c3)
  ```

  For a table with many columns, however, this syntax might be cumbersome and prone to error. An alternate method of loading the file is to use the `generatedmissing` file type modifier as follows:

  ```
  DB2 LOAD FROM load.del of del MODIFIED BY generatedmissing
     REPLACE INTO table1
  ```

- The `generatedignore` modifier is in some ways the opposite of the `generatedmissing` modifier: it indicates to the load utility that even though the input data file contains data for all generated columns present in the target table, the data should be ignored, and the computed values should be loaded into each generated column. For example, if you want to load TABLE1, as defined above, from a data file (`load.del`) containing the following data:

  ```
  1, 5, 10, 15, J
  2, 6, 11, 16, K
  3, 7, 12, 17, I
  ```

The user-supplied, non-NULL values of 10, 11, and 12 (for g1), and 15, 16, and 17 (for g2) result in the row being rejected (SQL3550W). To avoid this, the user could issue the following LOAD command:

```
DB2 LOAD FROM load.del of del method P(1, 2, 5)
   REPLACE INTO table1 (c1, c2, c3)
```

Again, this approach might be cumbersome and prone to error if the table has many columns. The generatedignore modifier simplifies the syntax as follows:

```
DB2 LOAD FROM load.del of del MODIFIED BY generatedignore
   REPLACE INTO table1
```

- The generatedoverride modifier is used for loading user-supplied values into a table with generated columns. This can be useful when migrating data from another database system, or when loading a table from data that was recovered using the RECOVER DROPPED TABLE option of the ROLLFORWARD DATABASE command. When this modifier is used, any rows with no data (or NULL data) for non-nullable generated columns are rejected (SQL3116W).

  When this modifier is used, the table is placed in the set integrity pending state after the load operation. To take the table out of set integrity pending state without verifying the user-supplied values, issue the following command:

  ```
  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:

  ```
  SET INTEGRITY FOR table-name IMMEDIATE CHECKED.
  ```

If a generated column is in any of the partitioning, dimension or distribution keys, the generatedoverride modifier is ignored and the load utility generates values as if the generatedignore modifier is specified. Loading an incorrect generated column value in this case can place the record in the wrong physical location, such as the wrong data partition, MDC block or database partition. For example, once a record is on a wrong data partition, the set integrity operation has to move it to a different physical location, which cannot be accomplished during online set integrity operations.

For these generated columns, the data for the dependent columns must appear within the first 32KB of data for each row being loaded.

For example, consider a table created with the following SQL statement:

```
CREATE TABLE table1 (c1 INT, c2 INT, g1 INT GENERATED ALWAYS AS (c1 + c2))
  DISTRIBUTE BY hash (g1)
```

In order to successfully load data into this table, all of the data for columns c1 and c2 must be located within the first 32KB of each row being loaded. Any row that does not satisfy this restriction is rejected.

Note: There is one case where load does NOT support generating column values: that is when one of the generated column expressions contains a user-defined function that is FENCED. If you attempt to load into such a table the load utility will fail. However, you can provide your own values for these types of generated columns by using the generatedoverride file type modifier of the load utility.

Related concepts:

- "Generated Columns" in *Developing SQL and External Routines*

Checking for integrity violations following a load operation

Following a load operation, the loaded table might be in set integrity pending state in either READ or NO ACCESS mode if any of the following conditions exist:

- The table has table check constraints or referential integrity constraints defined on it.
- The table has generated columns and a V7 or earlier client was used to initiate the load operation.
- The table has descendent immediate materialized query tables or descendent immediate staging tables referencing it.
- The table is a staging table or a materialized query table.

The `STATUS` flag of the `SYSCAT.TABLES` entry corresponding to the loaded table indicates the set integrity pending state of the table. For the loaded table to be fully usable, the `STATUS` must have a value of `N` and the `ACCESS MODE` must have a value of `F`, indicating that the table is fully accessible and in normal state.

If the loaded table has descendent tables, the SET INTEGRITY PENDING CASCADE parameter can be specified to indicate whether or not the set integrity pending state of the loaded table should be immediately cascaded to the descendent tables.

If the loaded table has constraints as well as descendent foreign key tables, dependent materialized query tables and dependent staging tables, and if all of the tables are in normal state prior to the load operation, the following will result based on the load parameters specified:

INSERT, ALLOW READ ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
The loaded table, its dependent materialized query tables and dependent staging tables are placed in set integrity pending state with read access.

INSERT, ALLOW READ ACCESS, and SET INTEGRITY PENDING CASCADE DEFERRED
Only the loaded table is placed in set integrity pending with read access. Descendent foreign key tables, descendent materialized query tables and descendent staging tables remain in their original states.

INSERT, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
The loaded table, its dependent materialized query tables and dependent staging tables are placed in set integrity pending state with no access.

INSERT or REPLACE, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE DEFERRED
Only the loaded table is placed in set integrity pending state with no access. Descendent foreign key tables, descendent immediate materialized query tables and descendent immediate staging tables remain in their original states.

REPLACE, ALLOW NO ACCESS, and SET INTEGRITY PENDING CASCADE IMMEDIATE
The table and all its descendent foreign key tables, descendent immediate materialized query tables, and descendent immediate staging tables are placed in set integrity pending state with no access.

Note: Specifying the ALLOW READ ACCESS option in a load replace operation results in an error.

To remove the set integrity pending state, use the SET INTEGRITY statement. The SET INTEGRITY statement checks a table for constraints violations, and takes the table out of set integrity pending state. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement can be used to incrementally process the constraints (that is, it checks only the appended portion of the table for constraints violations). For example:

```
db2 load from infile1.ixf of ixf insert into table1
db2 set integrity for table1 immediate checked
```

Only the appended portion of TABLE1 is checked for constraint violations. Checking only the appended portion for constraints violations is faster than checking the entire table, especially in the case of a large table with small amounts of appended data.

If a table is loaded with the SET INTEGRITY PENDING CASCADE DEFERRED option specified, and the SET INTEGRITY statement is used to check for integrity violations, the descendent tables are placed in set integrity pending state with no access. To take the tables out of this state, you must issue an explicit request.

If a table with dependent materialized query tables or dependent staging tables is loaded using the INSERT option, and the SET INTEGRITY statement is used to check for integrity violations, the table is taken out of set integrity pending state and placed in No Data Movement state. This is done to facilitate the subsequent incremental refreshes of the dependent materialized query tables and the incremental propagation of the dependent staging tables. In the No Data Movement state, operations that might cause the movement of rows within the table are not allowed.

You can override the No Data Movement state by specifying the FULL ACCESS option when you issue the SET INTEGRITY statement. The table is fully accessible, however a full re-computation of the dependent materialized query tables takes place in subsequent REFRESH TABLE statements and the dependent staging tables are forced into an incomplete state.

If the ALLOW READ ACCESS option is specified for a load operation, the table remains in read access state until the SET INTEGRITY statement is used to check for constraints violations. Applications can query the table for data that existed prior to the load operation once it has been committed, but will not be able to view the newly loaded data until the SET INTEGRITY statement is issued.

Several load operations can take place on a table before checking for constraints violations. If all of the load operations are completed in ALLOW READ ACCESS mode, only the data that existed in the table prior to the first load operation is available for queries.

One or more tables can be checked in a single invocation of this statement. If a dependent table is to be checked on its own, the parent table can not be in set integrity pending state. Otherwise, both the parent table and the dependent table must be checked at the same time. In the case of a referential integrity cycle, all the tables involved in the cycle must be included in a single invocation of the SET INTEGRITY statement. It might be convenient to check the parent table for constraints violations while a dependent table is being loaded. This can only occur if the two tables are not in the same table space.

When issuing the SET INTEGRITY statement, you can specify the INCREMENTAL option to explicitly request incremental processing. In most cases, this option is not needed, because the DB2 database selects incremental processing. If incremental processing is not possible, full processing is used automatically. When the INCREMENTAL option is specified, but incremental processing is not possible, an error is returned if:

- New constraints are added to the table while it is in set integrity pending state.
- A load replace operation takes place, or the NOT LOGGED INITIALLY WITH EMPTY TABLE option is activated, after the last integrity check on the table.
- A parent table is load replaced or checked for integrity non-incrementally.
- The table is in set integrity pending state before migration. Full processing is required the first time the table is checked for integrity after migration.
- The table space containing the table or its parent is rolled forward to a point in time and the table and its parent reside in different table spaces.

If a table has one or more `W` values in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and if the NOT INCREMENTAL option is not specified in the SET INTEGRITY statement, the table is incrementally processed and the CONST_CHECKED column of SYSCAT.TABLES is marked as `U` to indicate that not all data has been verified by the system.

The SET INTEGRITY statement does not activate any DELETE triggers as a result of deleting rows that violate constraints, but once the table is removed from set integrity pending state, triggers are active. Thus, if you correct data and insert rows from the exception table into the loaded table, any INSERT triggers defined on the table are activated. The implications of this should be considered. One option is to drop the INSERT trigger, insert rows from the exception table, and then recreate the INSERT trigger.

Related concepts:

- "Load exception table" on page 200
- "Pending states after a load operation" on page 206
- "Read access load operations" on page 113

Related reference:

- "SET INTEGRITY statement" in *SQL Reference, Volume 2*
- "Exception tables" in *SQL Reference, Volume 1*

Refreshing dependent immediate materialized query tables

If the underlying table of an immediate refresh materialized query table is loaded using the INSERT option, executing the SET INTEGRITY statement on the dependent materialized query tables defined with REFRESH IMMEDIATE will result in an incremental refresh of the materialized query table. During an incremental refresh, the rows corresponding to the appended rows in the underlying tables are updated and inserted into the materialized query tables. Incremental refresh is faster in the case of large underlying tables with small amounts of appended data. There are cases in which incremental refresh is not allowed, and full refresh (that is, recomputation of the materialized query table definition query) will be used.

When the INCREMENTAL option is specified, but incremental processing of the materialized query table is not possible, an error is returned if:

- A load replace operation has taken place into an underlying table of the materialized query table or the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated since the last integrity check on the underlying table.
- The materialized query table has been loaded (in either REPLACE or INSERT mode).
- An underlying table has been taken out of Set Integrity Pending state before the materialized query table is refreshed by using the FULL ACCESS option during integrity checking.
- An underlying table of the materialized query table has been checked for integrity non-incrementally.
- The materialized query table was in Set Integrity Pending state before migration.
- The table space containing the materialized query table or its underlying table has been rolled forward to a point in time, and the materialized query table and its underlying table reside in different table spaces.

If the materialized query table has one or more `W` values in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and if the NOT INCREMENTAL option is not specified in the SET INTEGRITY statement, the table will be incrementally refreshed and the CONST_CHECKED column of SYSCAT.TABLES will be marked `U` to indicate that not all data has been verified by the system.

The following example illustrates a load insert operation into the underlying table UTI of the materialized query table AST1. UT1 will be checked for data integrity and will be placed in no data movement mode. UT1 will be put back into full access state once the incremental refresh of AST1 is complete. In this scenario, both the integrity checking for UT1 and the refreshing of AST1 will be processed incrementally.

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1 IMMEDIATE CHECKED;
REFRESH TABLE AST1;
```

Related concepts:

- "Checking for integrity violations following a load operation" on page 121

Propagating dependent immediate staging tables

If the table being loaded is an underlying table of a staging table with the immediate propagate attribute, and if the load operation is done in insert mode, the subsequent propagation into the dependent immediate staging tables will be incremental.

During incremental propagation, the rows corresponding to the appended rows in the underlying tables are appended into the staging tables. Incremental propagation is faster in the case of large underlying tables with small amounts of appended data. Performance will also be improved if the staging table is used to refresh its dependent deferred materialized query table. There are cases in which incremental propagation is not allowed, and the staging table will be marked incomplete. That is, the staging byte of the CONST_CHECKED column will have a value of F. In this state, the staging table can not be used to refresh its dependent deferred materialized query table, and a full refresh will be required in the materialized query table maintenance process.

If a table is in incomplete state and the INCREMENTAL option has been specified, but incremental propagation of the table is not possible, an error is returned. If any of the following have taken place, the system will turn off immediate data propagation and set the table state to incomplete:

- A load replace operation has taken place on an underlying table of the staging table, or the NOT LOGGED INITIALLY WITH EMPTY TABLE option has been activated after the last integrity check on the underlying table.
- The dependent materialized query table of the staging table, or the staging table has been loaded in REPLACE or INSERT mode.
- An underlying table has been taken out of Set Integrity Pending state before the staging table has been propagated by using the FULL ACCESS option during integrity checking.
- An underlying table of the staging table has been checked for integrity non-incrementally.
- The table space containing the staging table or its underlying table has been rolled forward to a point in time, and the staging table and its underlying table reside in different table spaces.

If the staging table has a `W` value in the CONST_CHECKED column of the SYSCAT.TABLES catalog, and the NOT INCREMENTAL option is not specified, incremental propagation to the staging table takes place and the CONST_CHECKED column of SYSCAT.TABLES will be marked as `U` to indicate that not all data has been verified by the system.

The following example illustrates a load insert operation into the underlying table UT1 of staging table G1 and its dependent deferred materialized query table AST1. In this scenario, both the integrity checking for UT1 and the refreshing of AST1 will be processed incrementally:

```
LOAD FROM IMTFILE1.IXF of IXF INSERT INTO UT1;
LOAD FROM IMTFILE2.IXF of IXF INSERT INTO UT1;
SET INTEGRITY FOR UT1,G1 IMMEDIATE CHECKED;

REFRESH TABLE AST1 INCREMENTAL;
```

Related concepts:

- "Checking for integrity violations following a load operation" on page 121

Multidimensional clustering considerations

The following restrictions apply when loading data into multi-dimensional clustering (MDC) tables:

- The SAVECOUNT option of the LOAD command is not supported.
- The TOTALFREESPACE file type modifier is not supported since these tables manage their own free space.
- The ANYORDER modifier is required for MDC tables. If a load is executed into an MDC table without the ANYORDER modifier, it will be explicitly enabled by the utility.

When using the LOAD command with MDC, violations of unique constraints will be handled as follows:

- If the table included a unique key prior to the load operation and duplicate records are loaded into the table, the original record will remain and the new records will be deleted during the delete phase.

- If the table did not include a unique key prior to the load operation and both a unique key and duplicate records are loaded into the table, only one of the records with the unique key will be loaded and the others will be deleted during the delete phase.

  Note: There is no explicit technique for determining which record will be loaded and which will be deleted.

Performance Considerations

To improve the performance of the load utility when loading MDC tables, the UTIL_HEAP_SZ database configuration parameter value should be increased. The mdc-load algorithm will perform significantly better when more memory is available to the utility.. This will reduce disk I/O during the clustering of data that is performed during the load phase. When the DATA BUFFER option of LOAD command is specified, its value should also be increased. If the LOAD command is being used to load several MDC tables concurrently, the UTIL_HEAP_SZ configuration parameter should be increased accordingly.

MDC load operations will always have a build phase since all MDC tables have block indexes.

During the load phase, extra logging for the maintenance of the block map will be performed. There are approximately two extra log records per extent allocated. To ensure good performance, the LOGBUFSZ database configuration parameter should be set to a value that takes this into account.

A system temporary table with an index is used to load data into MDC tables. The size of the table is proportional to the number of distinct cells loaded. The size of each row in the table is proportional to the size of the MDC dimension key. To minimize disk I/O caused by the manipulation of this table during a load operation, ensure that the buffer pool for the temporary table space is large enough.

Related concepts:
- "Optimizing load performance" on page 207
- "Multidimensional clustering tables" in *Administration Guide: Planning*

Load considerations for partitioned tables

All of the existing load features are supported when the target table is partitioned with the exception of the following general restrictions:
- Consistency points are not supported.
- Loading data into a subset of data partitions while the remaining data partitions remain fully online is not supported.
- The exception table used by a load operation cannot be partitioned.
- A unique index cannot be rebuilt when the load utility is running in insert mode or restart mode, and the load target table has any detached dependents.
- Similar to loading MDC tables, exact ordering of input data records is not preserved when loading partitioned tables. Ordering is only maintained within the cell or data partition.
- Load operations utilizing multiple formatters on each database partition only preserve approximate ordering of input records. Running a single formatter on

each database partition, groups the input records by cell or table partitioning key. To run a single formatter on each database partition, explicitly request CPU_PARALLELISM of 1.

General load behavior

The load utility inserts data records into the correct data partition. There is no requirement to use an external utility, such as a splitter, to partition the input data before loading.

The load utility does not access any detached or attached data partitions. Data is inserted into visible data partitions only. Visible data partitions are neither attached nor detached. In addition, a load replace operation does not truncate detached or attached data partitions. Since the load utility acquires locks on the catalog system tables, the load utility waits for any uncommitted ALTER TABLE transactions. Such transactions acquire an exclusive lock on the relevant rows in the catalog tables, and the exclusive lock must terminate before the load operation can proceed. This means that there can be no uncommitted ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION transactions while load operation is running. Any input source records destined for an attached or detached data partition are rejected, and can be retrieved from the exception table if one is specified. An informational message is written to the message file to indicate some of the target table data partitions were in an attached or detached state. Locks on the relevant catalog table rows corresponding to the target table prevent users from changing the partitioning of the target table by issuing any ALTER TABLE ...ATTACH, DETACH, or ADD PARTITION operations while the load utility is running.

Handling of invalid rows

When the load utility encounters a record that does not belong to any of the visible data partitions the record is rejected and the load utility continues processing. The number of records rejected because of the range constraint violation is not explicitly displayed, but is included in the overall number of rejected records. Rejecting a record because of the range violation does not increase the number of row warnings. A single message (SQL0327N) is written to the load utility message file indicating that range violations are found, but no per-record messages are logged. The load utility offers an option to have otherwise valid rows that were rejected because of a range constraint violation, inserted into the exception table. In addition to all columns of the target table, the exception table includes columns describing the type of violation that had occurred for a particular row. Rows containing invalid data, including data that cannot be partitioned, are written to the dump file. Because exception table inserts are expensive, you can control which constraint violations are inserted into the exception table. If you do not specify the exception table, or opt not to have range violating rows inserted into the exception table, information about rows violating the range constraint are lost.

History file

If the target table is partitioned, the corresponding history file entry does not include a list of the table spaces spanned by the target table. A different operation granularity identifier ('R' instead of 'T') indicates that a load operation ran against a partitioned table.

Terminating a load operation

Terminating a load replace completely truncates all visible data partitions, terminating a load insert truncates all visible data partitions to their lengths before the load. Indices are invalidated during a termination of an online load operation that failed in the load copy phase. Indices are also invalidated when terminating an offline load operation that touched the index (It is invalidated because the indexing mode is rebuild, or a key was inserted during incremental maintenance leaving the index in an inconsistent state). Loading data into multiple targets does not have any effect on load recovery operations except for the inability to restart the load operation from a consistency point taken during the load phase In this case, the SAVECOUNT load option is ignored if the target table is partitioned. This behavior is consistent with loading data into a MDC target table.

Generated columns

If a generated column is in any of the partitioning, dimension or distribution keys, the GENERATEDOVERRIDE modifier is ignored and the load utility generates values as if the GENERATEDIGNORE modifier is specified. Loading an incorrect generated column value in this case can place the record in the wrong physical location, such as the wrong data partition, MDC block or database partition. For example, once a record is on a wrong data partition, set integrity has to move it to a different physical location, which cannot be accomplished during online set integrity operations.

Data availability

The current online load algorithm extends to partitioned tables. An online load (’ALLOW READ ACCESS) specified on the LOAD command allows concurrent readers to access the whole table, including both loading and non-loading data partitions.

Data partition states

After a successful load, visible data partitions might change to either or both SET INTEGRITY PENDING or READ ONLY state, under certain conditions. Data partitions might be placed in these states if there are constraints on the table which the load operation cannot maintain. Such constraints might include check contraints and detached materialized query tables. A failed load operation leaves all visible data partitions in the LOAD PENDING state.

Error isolation

Error isolation at the data partition level is not supported. Isolating the errors means continuing a load on data partitions that did not run into an error and stopping on data partitions that did run into an error. Errors can be isolated between different database partitions, but the load utility cannot commit transactions on a subset of visible data partitions and rollback the remaining visible data partitions.

Other considerations
- Incremental indexing is not supported if any of the indexes are marked invalid. An index is considered invalid if it requires a rebuild or if detached dependents require validation with the SET INTEGRITY statement.
- Loading into tables partitioned using any combination of partitioned by range, distributed by hash or organized by dimension algorithms is also supported.

- For log records which include the list of object and table space IDs affected by the load, the size of these log records (LOAD START and COMMIT (PENDING LIST)) could grow considerably and hence reduce the amount of active log space available to other applications.
- When a table is both partitioned and distributed, a partitioned database load might not affect all database partitions. Only the objects on the output database partitions are changed.
- During a load operation, memory consumption for partitioned tables increases with the number of tables. Note, that the total increase is not linear as only a small percentage of the overall memory requirement is proportional to the number of data partitions.

Related concepts:
- "Load in a partitioned database environment - overview" on page 217
- "Partitioned tables" in *Administration Guide: Planning*
- "Data partitions" in *Administration Guide: Planning*
- "Load overview" on page 102

Related tasks:
- "Loading data into a table using the Load wizard" in *Administration Guide: Implementation*
- "Loading data" on page 110

Related reference:
- "LOAD " on page 132
- "db2Load - Load data into a table" on page 161
- "Load - CLP examples" on page 212
- "Restrictions on native XML data store" in *XML Guide*

Restarting an interrupted load operation

If the load utility cannot start because of a user error, such as a nonexistent data file or invalid column names, it will terminate and leave the table in a normal state.

If a failure occurs while loading data, you can restart the load operation from the last consistency point (using the RESTART option), or reload the entire table (using the REPLACE option). Specify the same parameters as in the previous invocation, so that the utility can find the necessary temporary files. Because the SAVECOUNT parameter is not supported for multi-dimensional clustering (MDC) tables, a load restart will only take place at the beginning of the load, build, or delete phase.

Note: A load operation that specified the ALLOW READ ACCESS option can be restarted using either the ALLOW READ ACCESS option or the ALLOW NO ACCESS option. Conversely, a load operation that specified the ALLOW NO ACCESS option can not be restarted using the ALLOW READ ACCESS option.

Restarting or Terminating an Allow Read Access Load Operation

A aborted load operation that specified the ALLOW READ ACCESS option might also be restarted or terminated using the ALLOW READ ACCESS option. This will allow other applications to query the table data while the terminate or restart operation is in progress. As with a load operation in ALLOW READ ACCESS mode, the table is locked exclusively prior to the data being committed.

If the index object is unavailable or marked invalid, a load restart or terminate operation in ALLOW READ ACCESS mode will not be permitted.

If the original load operation was aborted in the index copy phase, a restart operation in the ALLOW READ ACCESS mode is not permitted because the index might be corrupted.

If a load operation in ALLOW READ ACCESS mode was aborted in the load phase, it will restart in the load phase. If it was aborted in any phase other than the load phase, it will restart in the build phase. If the original load operation was in ALLOW NO ACCESS mode, a restart operation might occur in the delete phase if the original load operation reached that point and the index is valid. If the index is marked invalid, the load utility will restart the load operation from the build phase.

Note: All load restart operations will choose the REBUILD indexing mode even if the INDEXING MODE INCREMENTAL option is specified.

Issuing a LOAD TERMINATE command will generally cause the aborted load operation to be rolled back with minimal delay. However, when issuing a LOAD TERMINATE command for a load operation where ALLOW READ ACCESS and INDEXING MODE INCREMENTAL are specified, there might be a delay while the load utility scans the indexes and corrects any inconsistencies. The length of this delay will depend on the size of the indexes and will occur whether or not the ALLOW READ ACCESS option is specified for the load terminate operation. The delay will not occur if the original load operation failed prior to the build phase.

Note: The delay resulting from corrections to inconsistencies in the index will be considerably less than the delay caused by marking the indexes as invalid and rebuilding them.

A load restart operation cannot be undertaken on a table that is in the not load restartable table state. The table can be placed in the not load restartable table state during a rollforward operation. This can occur if you roll forward to a point in time that is prior to the end of a load operation, or if you roll forward through an aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

Related concepts:

- "Restarting or terminating a load operation in a partitioned database environment" on page 227
- "Table locking, table states and table space states" on page 203
- "Load dump file" on page 201
- "Load exception table" on page 200
- "Load overview" on page 102

Related reference:
- "Load configuration options for partitioned database environments" on page 229

Recovering data with the load copy location file

The DB2LOADREC registry variable is used to identify the file with the load copy location information. This file is used during rollforward recovery to locate the load copy. It has information about:
- Media type
- Number of media devices to be used
- Location of the load copy generated during a table load operation
- File name of the load copy, if applicable

If the location file does not exist, or no matching entry is found in the file, the information from the log record is used.

The information in the file might be overwritten before rollforward recovery takes place.

Notes:
1. In a multi-partition database, the DB2LOADREC registry variable must be set for all the database partition servers using the **db2set** command.
2. In a multi-partition database, the load copy file must exist at each database partition server, and the file name (including the path) must be the same.
3. If an entry in the file identified by the DB2LOADREC registry variable is not valid, the old load copy location file is used to provide information to replace the invalid entry.

The following information is provided in the location file. The first five parameters must have valid values, and are used to identify the load copy. The entire structure is repeated for each load copy recorded. For example:

```
TIMestamp      19950725182542         * Time stamp generated at load time
DBPartition    0                      * DB Partition number (OPTIONAL)
SCHema         PAYROLL                * Schema of table loaded
TABlename      EMPLOYEES              * Table name
DATabasename   DBT                    * Database name
DB2instance    toronto                * DB2INSTANCE
BUFfernumber   NULL                   * Number of buffers to be used for
                                        recovery
SESsionnumber  NULL                   * Number of sessions to be used for
                                        recovery
TYPeofmedia    L                      * Type of media - L for local device
                                                        A for TSM
                                                        O for other vendors
LOCationnumber 3                      * Number of locations
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
DBP            18
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            toronto
BUF            NULL
SES            NULL
TYP            A
TIM            19940325192054
```

```
SCH          PAYROLL
TAB          DEPT
DAT          DBT
DB2          toronto
BUF          NULL
SES          NULL
TYP          O
SHRlib       /@sys/lib/backup_vendor.a
```

Notes:

1. The first three characters in each keyword are significant. All keywords are required in the specified order. Blank lines are not accepted.
2. The time stamp is in the form *yyyymmddhhmmss*.
3. All fields are mandatory, except for BUF and SES (which can be NULL), and DBP (which can be missing from the list).. If SES is NULL, the value specified by the *numloadrecses* configuration parameter is used. If BUF is NULL, the default value is SES+2.
4. If even one of the entries in the location file is invalid, the previous load copy location file is used to provide those values.
5. The media type can be local device (`L` for tape, disk or diskettes), TSM (`A`), or other vendor (`O`). If the type is `L`, the number of locations, followed by the location entries, is required. If the type is `A`, no further input is required. If the type is `O`, the shared library name is required.
6. The SHRlib parameter points to a library that has a function to store the load copy data.
7. If you invoke a load operation, specifying the COPY NO or the NONRECOVERABLE option, and do not take a backup copy of the database or affected table spaces after the operation completes, you cannot restore the database or table spaces to a point in time that follows the load operation. That is, you cannot use rollforward recovery to recreate the database or table spaces to the state they were in following the load operation. You can only restore the database or table spaces to a point in time that precedes the load operation.

If you want to use a particular load copy, you can use the recovery history file for the database to determine the time stamp for that specific load operation. In a multi-partition database, the recovery history file is local to each database partition.

Related reference:

- "Tivoli Storage Manager" in *Data Recovery and High Availability Guide and Reference*

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

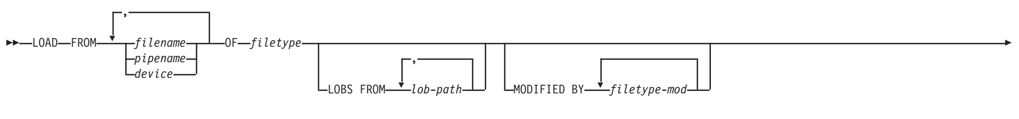
  If the session authorization id does not hold such a security label then the load fails and an error (SQLSTATE 5U014) is returned. This security label is used to protect a loaded row if the session authorization ID's LBAC credentials do not allow it to write to the security label that protects that row 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.

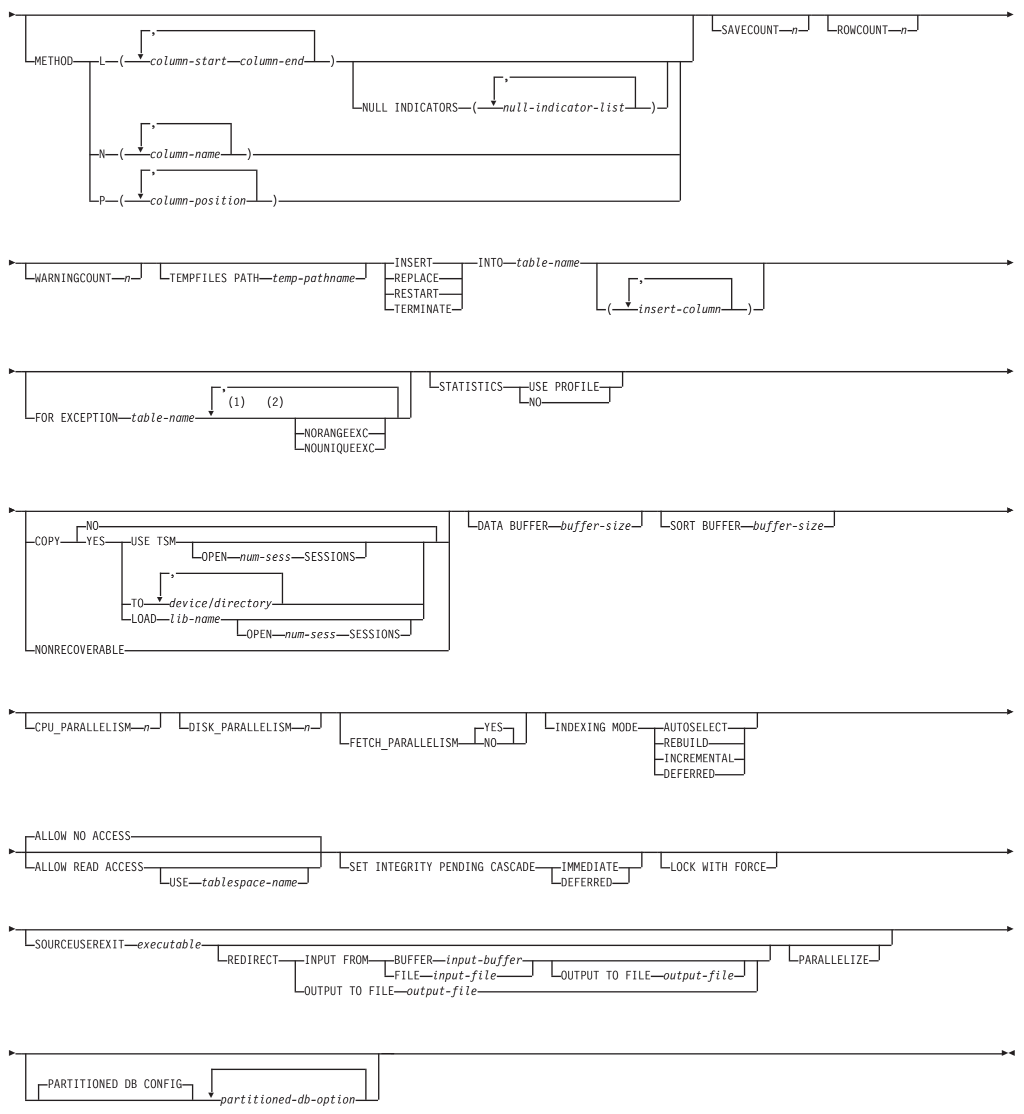
Since all load processes (and all DB2 server processes, in general) are owned by the instance owner, and all of these processes use the identification of the instance owner to access needed files, the instance owner must have read access to input data files. These input data files must be readable by 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:





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" on page 102
- "Privileges, authorities, and authorizations required to use Load" on page 109

Related tasks:
- "Loading data" on page 110

Related reference:
- "QUIESCE TABLESPACES FOR TABLE command" in *Command Reference*
- "LOAD command using the ADMIN_CMD procedure" on page 145
- "Load - CLP examples" on page 212
- "Load configuration options for partitioned database environments" on page 229

LOAD command using the ADMIN_CMD procedure

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─┬┴─OF─filetype──┬──────────────────────┬──┬────────────────────────────┬──────────►
               ├─pipename─┤               │           ┌─,──────┐ │  │              ┌────────────┐ │
               └─device───┘               │           ▼        │ │  │              ▼            │ │
                                          └─LOBS FROM───lob-path─┴┘  └─MODIFIED BY───filetype-mod─┴┘
```

```
►─┬─────────────────────────────────────────────────────────────────────────┬──┬──────────────┬──┬─────────────┬──►
  │              ┌─,───────────────────────┐                                │  └─SAVECOUNT──n─┘  └─ROWCOUNT──n─┘
  └─METHOD──┬─L──(───▼─column-start──column-end─┴──)──┬──────────────────────────────────────────┬─┬─┘
            │                                         │                   ┌─,───────────────────┐ │ │
            │                                         └─NULL INDICATORS──(───▼─null-indicator-list─┴──)─┘ │
            │      ┌─,───────────┐                                                                  │
            ├─N──(───▼─column-name─┴──)─────────────────────────────────────────────────────────────┤
            │      ┌─,───────────────┐                                                              │
            └─P──(───▼─column-position─┴──)─────────────────────────────────────────────────────────┘

►─┬──────────────────┬──┬──────────────────────────────┬──┬─INSERT────┬──INTO──table-name──┬────────────────────────┬──►
  └─WARNINGCOUNT──n─┘  └─TEMPFILES PATH──temp-pathname─┘  ├─REPLACE───┤                    │   ┌─,─────────────┐    │
                                                          ├─RESTART───┤                    └─(───▼─insert-column─┴──)─┘
                                                          └─TERMINATE─┘

►─┬─────────────────────────────────────────────────────┬──┬──────────────────────────────────┬──►
  │                                 ┌──────────────────┐ │  └─STATISTICS──┬─USE PROFILE─┬─────┘
  │                                 ▼ (1)   (2)         │ │                └─NO──────────┘
  └─FOR EXCEPTION──table-name──────┴──┬─NORANGEEXC──┬──┴─┘
                                      └─NOUNIQUEEXC─┘

►─┬──────────────────────────────────────────────────────────────┬──┬────────────────────────────┬──┬────────────────────────────┬──►
  │       ┌─NO───────────────────────────────────────────────┐   │  └─DATA BUFFER──buffer-size─┘  └─SORT BUFFER──buffer-size─┘
  ├─COPY──┴─YES──┬─USE TSM──┬─────────────────────────────┬─┬─┤
  │              │          └─OPEN──num-sess──SESSIONS─┘   │ │
  │              │       ┌─,────────────────┐              │ │
  │              ├─TO────▼─device/directory─┴──────────────┤ │
  │              └─LOAD──lib-name──┬──────────────────────────┬┘ │
  │                                └─OPEN──num-sess──SESSIONS─┘  │
  └─NONRECOVERABLE──────────────────────────────────────────────┘

►─┬────────────────────┬──┬─────────────────────┬──┬─────────────────────────────────┬──┬────────────────────────────────────┬──►
  └─CPU_PARALLELISM──n─┘  └─DISK_PARALLELISM──n─┘  │                    ┌─YES─┐      │  └─INDEXING MODE──┬─AUTOSELECT──┬────┘
                                                   └─FETCH_PARALLELISM──┴─NO──┴──────┘                    ├─REBUILD─────┤
                                                                                                           ├─INCREMENTAL─┤
                                                                                                           └─DEFERRED────┘

  ┌─ALLOW NO ACCESS───────────────────────────────┐
►─┴─ALLOW READ ACCESS──┬──────────────────────────┬─┴──┬──────────────────────────────────────────────────┬──┬─────────────────┬──►
                       └─USE──tablespace-name─────┘    └─SET INTEGRITY PENDING CASCADE──┬─IMMEDIATE─┬─────┘  └─LOCK WITH FORCE─┘
                                                                                        └─DEFERRED──┘

►─┬───────────────────────────────────────────────────────────────────────────────────────────────────────────────┬──►
  └─SOURCEUSEREXIT──executable──┬────────────────────────────────────────────────────────────────┬──┬─────────────┬┘
                                └─REDIRECT──┬─INPUT FROM──┬─BUFFER──input-buffer─┬──┬────────────────────────────┬┬┘  └─PARALLELIZE─┘
                                            │             └─FILE──input-file─────┘  └─OUTPUT TO FILE──output-file─┘│
                                            └─OUTPUT TO FILE──output-file──────────────────────────────────────────┘

►─┬───────────────────────────────────────────────────────────────┬──────────────────────────────────────────►◄
  │                        ┌───────────────────────┐              │
  └─┬─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:

- "Privileges, authorities, and authorizations required to use Load" on page 109
- "Load overview" on page 102

Related reference:

- "ADMIN_GET_MSGS table function – Retrieve messages generated by a data movement utility that is executed through the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "ADMIN_REMOVE_MSGS procedure – Clean up messages generated by a data movement utility that is executed through the ADMIN_CMD procedure" in *Administrative SQL Routines and Views*
- "EXPORT command using the ADMIN_CMD procedure" on page 15
- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "Load configuration options for partitioned database environments" on page 229
- "db2pd - Monitor and troubleshoot DB2 database command" in *Command Reference*

LOAD QUERY

Checks the status of a load operation during processing and returns the table state. If a load is not processing, then the table state alone is returned. A connection to the same database, and a separate CLP session are also required to successfully invoke this command. It can be used either by local or remote users.

Authorization:

None

Required connection:

Database

Command syntax:

```
►►──LOAD QUERY──TABLE──table-name──┬──────────────────────┬──┬─────────────┬──────►
                                   └─TO──local-message-file─┘  ├─NOSUMMARY───┤
                                                               └─SUMMARYONLY─┘

►──┬───────────┬──────────────────────────────────────────────────────────────►◄
   └─SHOWDELTA─┘
```

Command parameters:

NOSUMMARY
: Specifies that no load summary information (rows read, rows skipped, rows loaded, rows rejected, rows deleted, rows committed, and number of warnings) is to be reported.

SHOWDELTA
: Specifies that only new information (pertaining to load events that have occurred since the last invocation of the **LOAD QUERY** command) is to be reported.

SUMMARYONLY
: Specifies that only load summary information is to be reported.

TABLE table-name
: Specifies the name of the table into which data is currently being loaded. If an unqualified table name is specified, the table will be qualified with the CURRENT SCHEMA.

TO local-message-file
: Specifies the destination for warning and error messages that occur during the load operation. This file cannot be the *message-file* specified for the LOAD command. If the file already exists, all messages that the load utility has generated are appended to it.

Examples:

A user loading a large amount of data into the STAFF table wants to check the status of the load operation. The user can specify:

```
   db2 connect to <database>
   db2 load query table staff to /u/mydir/staff.tempmsg
```

The output file `/u/mydir/staff.tempmsg` might look like the following:

```
SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL3109N  The utility is beginning to load data from file
"/u/mydir/data/staffbig.del"

SQL3500W  The utility is beginning the "LOAD" phase at time "03-21-2002
11:31:16.597045".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "104416".
```

```
SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "205757".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "307098".

SQL3520W  Load Consistency Point was successful.

SQL3519W  Begin Load Consistency Point. Input record count = "408439".

SQL3520W  Load Consistency Point was successful.

SQL3532I  The Load utility is currently in the "LOAD" phase.


Number of rows read         = 453376
Number of rows skipped      = 0
Number of rows loaded       = 453376
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 408439
Number of warnings          = 0

Tablestate:
  Load in Progress
```

Usage Notes:

In addition to locks, the load utility uses table states to control access to the table. The **LOAD QUERY** command can be used to determine the table state; **LOAD QUERY** can be used on tables that are not currently being loaded. For a partitioned table, the state reported is the most restrictive of the corresponding visible data partition states. For example, if a single data partition is in the READ ACCESS state and all other data partitions are in NORMAL state, the load query operation returns the READ ACCESS state. A load operation will not leave a subset of data partitions in a state different from the rest of the table. The table states described by **LOAD QUERY** are as follows:

Normal
: No table states affect the table.

Set Integrity Pending
: The table has constraints which have not yet been verified. Use the SET INTEGRITY statement to take the table out of Set Integrity Pending state. The load utility places a table in Set Integrity Pending state when it begins a load operation on a table with constraints.

Load in Progress
: There is a load operation in progress on this table.

Load Pending
: A load operation has been active on this table but has been aborted before the data could be committed. Issue a LOAD TERMINATE, LOAD RESTART, or LOAD REPLACE command to bring the table out of this state.

Read Access Only
: The table data is available for read access queries. Load operations using the ALLOW READ ACCESS option place the table in read access only state.

Reorg Pending
A reorg recommended `ALTER TABLE` statement has been executed on the table. A classic reorg must be performed before the table is accessable again.

Unavailable
The table is unavailable. The table can only be dropped or restored from a backup. Rolling forward through a non-recoverable load operation will place a table in the unavailable state.

Not Load Restartable
The table is in a partially loaded state that will not allow a load restart operation. The table will also be in load pending state. Issue a LOAD TERMINATE or a LOAD REPLACE command to bring the table out of the not load restartable state. A table is placed in not load restartable state when a rollforward operation is performed after a failed load operation that has not been successfully restarted or terminated, or when a restore operation is performed from an online backup that was taken while the table was in load in progress or load pending state. In either case, the information required for a load restart operation is unreliable, and the not load restartable state prevents a load restart operation from taking place.

Unknown
The **LOAD QUERY** command is unable determine the table state.

The progress of a load operation can also be monitored with the LIST UTILITIES command.

Related concepts:
- "Load overview" on page 102
- "Monitoring a load operation in a partitioned database environment using the LOAD QUERY command" on page 225
- "Table locking, table states and table space states" on page 203

Related reference:
- "LIST UTILITIES command" in *Command Reference*

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" on page 110

Related reference:
- "LOAD " on page 132
- "sqldcol data structure" in *Administrative API Reference*
- "sqlu_media_list data structure" in *Administrative API Reference*
- "db2LoadQuery - Get the status of a load operation" on page 181
- "db2Export - Export data from a database" on page 19
- "db2Import - Import data into a table, hierarchy, nickname or view" on page 73

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++)"

db2LoadQuery - Get the status of a load operation

Checks the status of a load operation during processing.

Authorization:

None

Required connection:

Database

API include file:

```
db2ApiDf.h
```

API and data structure syntax:

```
SQL_API_RC SQL_API_FN
  db2LoadQuery (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2LoadQueryStruct
{
   db2Uint32 iStringType;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct *poOutputStruct;
   char *piLocalMessageFile;
} db2LoadQueryStruct;
```

```
typedef SQL_STRUCTURE db2LoadQueryOutputStruct
{
   db2Uint32 oRowsRead;
   db2Uint32 oRowsSkipped;
   db2Uint32 oRowsCommitted;
   db2Uint32 oRowsLoaded;
   db2Uint32 oRowsRejected;
   db2Uint32 oRowsDeleted;
   db2Uint32 oCurrentIndex;
   db2Uint32 oNumTotalIndexes;
   db2Uint32 oCurrentMPPNode;
   db2Uint32 oLoadRestarted;
   db2Uint32 oWhichPhase;
   db2Uint32 oWarningCount;
   db2Uint32 oTableState;
} db2LoadQueryOutputStruct;

typedef SQL_STRUCTURE db2LoadQueryOutputStruct64
{
   db2Uint64 oRowsRead;
   db2Uint64 oRowsSkipped;
   db2Uint64 oRowsCommitted;
   db2Uint64 oRowsLoaded;
   db2Uint64 oRowsRejected;
   db2Uint64 oRowsDeleted;
   db2Uint32 oCurrentIndex;
   db2Uint32 oNumTotalIndexes;
   db2Uint32 oCurrentMPPNode;
   db2Uint32 oLoadRestarted;
   db2Uint32 oWhichPhase;
   db2Uint32 oWarningCount;
   db2Uint32 oTableState;
} db2LoadQueryOutputStruct64;

typedef SQL_STRUCTURE db2LoadQueryStruct64
{
   db2Uint32 iStringType;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct64 *poOutputStruct;
   char *piLocalMessageFile;
} db2LoadQueryStruct64;

SQL_API_RC SQL_API_FN
  db2gLoadQuery (
   db2Uint32 versionNumber,
   void * pParmStruct,
   struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gLoadQueryStruct
{
   db2Uint32 iStringType;
   db2Uint32 iStringLen;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct *poOutputStruct;
   db2Uint32 iLocalMessageFileLen;
   char *piLocalMessageFile;
} db2gLoadQueryStruct;

typedef SQL_STRUCTURE db2gLoadQueryStru64
{
   db2Uint32 iStringType;
   db2Uint32 iStringLen;
   char *piString;
   db2Uint32 iShowLoadMessages;
   struct db2LoadQueryOutputStruct64 *poOutputStruct;
   db2Uint32 iLocalMessageFileLen;
   char *piLocalMessageFile;
} db2gLoadQueryStru64;
```

db2LoadQuery API parameters:

versionNumber
: Input. Specifies the version and release level of the structure passed in as the second parameter, pParmStruct.

pParmStruct
: Input. A pointer to the db2LoadQueryStruct structure.

pSqlca
: Output. A pointer to the sqlca structure.

db2LoadQueryStruct data structure parameters:

iStringType
: Input. Specifies a type for piString. Valid values (defined in db2ApiDf header file, located in the include directory) are:

  DB2LOADQUERY_TABLENAME
  : Specifies a table name for use by the db2LoadQuery API.

piString
: Input. Specifies a temporary files path name or a table name, depending on the value of iStringType.

iShowLoadMessages
: Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in db2ApiDf header file, located in the include directory) are:

  DB2LOADQUERY_SHOW_ALL_MSGS
  : Return all load messages.

  DB2LOADQUERY_SHOW_NO_MSGS
  : Return no load messages.

  DB2LOADQUERY_SHOW_NEW_MSGS
  : Return only messages that have been generated since the last call to this API.

poOutputStruct
: Output. A pointer to the db2LoadQueryOutputStruct structure, which contains load summary information. Set to NULL if a summary is not required.

piLocalMessageFile
: Input. Specifies the name of a local file to be used for output messages.

db2LoadQueryOutputStruct data structure parameters:

oRowsRead
: Output. Number of records read so far by the load utility.

oRowsSkipped
: Output. Number of records skipped before the load operation began.

oRowsCommitted
: Output. Number of rows committed to the target table so far.

oRowsLoaded
Output. Number of rows loaded into the target table so far.

oRowsRejected
Output. Number of rows rejected from the target table so far.

oRowsDeleted
Output. Number of rows deleted from the target table so far (during the delete phase).

oCurrentIndex
Output. Index currently being built (during the build phase).

oNumTotalIndexes
Output. Total number of indexes to be built (during the build phase).

oCurrentMPPNode
Output. Indicates which database partition server is being queried (for partitioned database environment mode only).

oLoadRestarted
Output. A flag whose value is TRUE if the load operation being queried is a load restart operation.

oWhichPhase
Output. Indicates the current phase of the load operation being queried. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_LOAD_PHASE
Load phase.

DB2LOADQUERY_BUILD_PHASE
Build phase.

DB2LOADQUERY_DELETE_PHASE
Delete phase.

DB2LOADQUERY_INDEXCOPY_PHASE
Index copy phase.

oWarningCount
Output. Total number of warnings returned so far.

oTableState
Output. The table states. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_NORMAL
No table states affect the table.

DB2LOADQUERY_SI_PENDING
The table has constraints and the constraints have yet to be verified. Use the SET INTEGRITY command to take the table out of the DB2LOADQUERY_SI_PENDING state. The load utility puts a table into the DB2LOADQUERY_SI_PENDING state when it begins a load on a table with constraints.

DB2LOADQUERY_LOAD_IN_PROGRESS
There is a load actively in progress on this table.

DB2LOADQUERY_LOAD_PENDING
A load has been active on this table but has been aborted before the load could commit. Issue a load terminate, a load restart, or a

load replace to bring the table out of the DB2LOADQUERY_LOAD_PENDING state.

DB2LOADQUERY_REORG_PENDING
A reorg recommended alter has been performed on this table. A classic reorg must be performed before the table will be accessible.

DB2LOADQUERY_READ_ACCESS
The table data is available for read access queries. Loads using the DB2LOADQUERY_READ_ACCESS option put the table into Read Access Only state.

DB2LOADQUERY_NOTAVAILABLE
The table is unavailable. The table may only be dropped or it may be restored from a backup. Rollforward through a non-recoverable load will put a table into the unavailable state.

DB2LOADQUERY_NO_LOAD_RESTART
The table is in a partially loaded state that will not allow a load restart. The table will also be in the Load Pending state. Issue a load terminate or a load replace to bring the table out of the No Load Restartable state. The table can be placed in the DB2LOADQUERY_NO_LOAD_RESTART state during a rollforward operation. This can occur if you rollforward to a point in time that is prior to the end of a load operation, or if you roll forward through an aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

DB2LOADQUERY_TYPE1_INDEXES
The table currently uses type-1 indexes. The indexes can be converted to type-2 using the CONVERT option when using the REORG utility on the indexes.

db2LoadQueryOutputStruct64 data structure parameters:

oRowsRead
Output. Number of records read so far by the load utility.

oRowsSkipped
Output. Number of records skipped before the load operation began.

oRowsCommitted
Output. Number of rows committed to the target table so far.

oRowsLoaded
Output. Number of rows loaded into the target table so far.

oRowsRejected
Output. Number of rows rejected from the target table so far.

oRowsDeleted
Output. Number of rows deleted from the target table so far (during the delete phase).

oCurrentIndex
Output. Index currently being built (during the build phase).

oNumTotalIndexes
Output. Total number of indexes to be built (during the build phase).

oCurrentMPPNode
Output. Indicates which database partition server is being queried (for partitioned database environment mode only).

oLoadRestarted
Output. A flag whose value is TRUE if the load operation being queried is a load restart operation.

oWhichPhase
Output. Indicates the current phase of the load operation being queried. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_LOAD_PHASE
Load phase.

DB2LOADQUERY_BUILD_PHASE
Build phase.

DB2LOADQUERY_DELETE_PHASE
Delete phase.

DB2LOADQUERY_INDEXCOPY_PHASE
Index copy phase.

oWarningCount
Output. Total number of warnings returned so far.

oTableState
Output. The table states. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_NORMAL
No table states affect the table.

DB2LOADQUERY_SI_PENDING
The table has constraints and the constraints have yet to be verified. Use the SET INTEGRITY command to take the table out of the DB2LOADQUERY_SI_PENDING state. The load utility puts a table into the DB2LOADQUERY_SI_PENDING state when it begins a load on a table with constraints.

DB2LOADQUERY_LOAD_IN_PROGRESS
There is a load actively in progress on this table.

DB2LOADQUERY_LOAD_PENDING
A load has been active on this table but has been aborted before the load could commit. Issue a load terminate, a load restart, or a load replace to bring the table out of the DB2LOADQUERY_LOAD_PENDING state.

DB2LOADQUERY_REORG_PENDING
A reorg recommended alter has been performed on this table. A classic reorg must be performed before the table will be accessible.

DB2LOADQUERY_READ_ACCESS
The table data is available for read access queries. Loads using the DB2LOADQUERY_READ_ACCESS option put the table into Read Access Only state.

DB2LOADQUERY_NOTAVAILABLE
The table is unavailable. The table may only be dropped or it may be restored from a backup. Rollforward through a non-recoverable load will put a table into the unavailable state.

DB2LOADQUERY_NO_LOAD_RESTART
The table is in a partially loaded state that will not allow a load

restart. The table will also be in the Load Pending state. Issue a load terminate or a load replace to bring the table out of the No Load Restartable state. The table can be placed in the DB2LOADQUERY_NO_LOAD_RESTART state during a rollforward operation. This can occur if you rollforward to a point in time that is prior to the end of a load operation, or if you roll forward through an aborted load operation but do not roll forward to the end of the load terminate or load restart operation.

DB2LOADQUERY_TYPE1_INDEXES
The table currently uses type-1 indexes. The indexes can be converted to type-2 using the CONVERT option when using the REORG utility on the indexes.

db2LoadQueryStruct64 data structure parameters:

iStringType
Input. Specifies a type for piString. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_TABLENAME
Specifies a table name for use by the db2LoadQuery API.

piString
Input. Specifies a temporary files path name or a table name, depending on the value of iStringType.

iShowLoadMessages
Input. Specifies the level of messages that are to be returned by the load utility. Valid values (defined in db2ApiDf header file, located in the include directory) are:

DB2LOADQUERY_SHOW_ALL_MSGS
Return all load messages.

DB2LOADQUERY_SHOW_NO_MSGS
Return no load messages.

DB2LOADQUERY_SHOW_NEW_MSGS
Return only messages that have been generated since the last call to this API.

poOutputStruct
Output. A pointer to the db2LoadQueryOutputStruct structure, which contains load summary information. Set to NULL if a summary is not required.

piLocalMessageFile
Input. Specifies the name of a local file to be used for output messages.

db2gLoadQueryStruct data structure specific parameters:

iStringLen
Input. Specifies the length in bytes of piString parameter.

iLocalMessageFileLen
Input. Specifies the length in bytes of piLocalMessageFile parameter.

db2gLoadQueryStru64 data structure specific parameters:

iStringLen
Input. Specifies the length in bytes of piString parameter.

iLocalMessageFileLen
Input. Specifies the length in bytes of piLocalMessageFile parameter.

Usage notes:

This API reads the status of the load operation on the table specified by piString, and writes the status to the file specified by pLocalMsgFileName.

Related concepts:

- "Monitoring a load operation in a partitioned database environment using the LOAD QUERY command" on page 225

Related reference:

- "LOAD QUERY " on page 158
- "SQLCA data structure" in *Administrative API Reference*
- "db2Load - Load data into a table" on page 161

Related samples:

- "loadqry.sqb -- Query the current status of a load (MF COBOL)"
- "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++)"

File type modifiers for the load utility

Table 12. 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.

Table 12. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description

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.
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.

Table 12. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
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.
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.

Table 12. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description

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.

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.

Table 12. Valid file type modifiers for the load utility: All file formats (continued)

Modifier	Description
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 13. 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.

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"`

Table 13. Valid file type modifiers for the load utility: ASCII file formats (ASC/DEL) (continued)

Modifier	Description

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 13. 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 13. 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`

File type modifiers for the load utility

Table 13. 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 14. 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 14. 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 15. 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 16. 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 17. 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 161
- "Delimiter restrictions for moving data" on page 259
- "LOAD " on page 132

Load exception table

The exception table is a user-created table that reflects the definition of the table being loaded, and includes some additional columns. It is specified by the FOR EXCEPTION clause on the LOAD command. An exception table might not contain an identity column or any other type of generated column. If an identity column is present in the primary table, the corresponding column in the exception table should only contain the column's type, length, and nullability attributes. The exception table cannot be partitioned, or have a unique index. However, the exception table is used to store copies of rows that violate unique index rules, range constraints and security policies.

A load exception table can be assigned to the table space where the table being loaded resides, or to another table space. In either case, the load exception table should be assigned to the same database partition group and have the same distribution key as the table being loaded.

A unique key is a key for which no two values are equal. The mechanism used to enforce this constraint is called a unique index. A primary key is a special case of a unique key. A table cannot have more than one primary key.

Note: Any rows rejected because of invalid data before the building of an index are not inserted into the exception table.

Rows are appended to existing information in the exception table; this can include invalid rows from previous load operations. If you want only the invalid rows from the current load operation, you must remove the existing rows before invoking the utility.

The exception table used with the load utility is identical to the exception tables used by the SET INTEGRITY statement.

An exception table should be used when loading data which has a unique index and the possibility of duplicate records. If an exception table is not specified, and duplicate records are found, the load operation continues, and only a warning message is issued about the deleted duplicate records. The records themselves are not logged.

After the load operation completes, information in the exception table can be used to correct data that is in error. The corrected data can then be inserted into the table.

Related reference:

- "Exception tables" in *SQL Reference, Volume 1*

Load dump file

Specifying the *dumpfile* modifier tells the load utility the name and the location of the exception file to which rejected rows are written. When running in a partitioned database environment, rows can be rejected either by the Partitioning Subagents or by the Loading Subagents. Because of this, the dump file name is given an extension that identifies the subagent type, as well as the database partition number where the exceptions were generated. For example, if you specified the following dump file value:

```
dumpfile = "/u/usrname/dumpit"
```

Then rows that were rejected by the Load Subagent on database partition five will be stored in a file named `/u/usrname/dumpit.load.005`, rows that were rejected by the Load Subagent on database partition two will be stored in a file named `/u/usrname/dumpit.load.002`, and rows that were rejected by the Partitioning Subagent on database partition two will be stored in a file named `/u/usrname/dumpit.part.002`, and so on.

For rows rejected by the Load Subagent, if the row is less than 32 768 bytes in length, the record is copied to the dump file in its entirety; if it is longer, a row fragment (including the final bytes of the record) is written to the file.

For rows rejected by the Partitioning Subagent, the entire row is copied to the dump file regardless of the record size.

Related reference:

- "LOAD " on page 132

Load temporary files

DB2 creates temporary binary files during load processing. These files are used for load crash recovery, load terminate operations, warning and error messages, and runtime control data. The temporary files are removed when the load operation completes without error.

The temporary files are written to a path that can be specified through the *temp-pathname* parameter of the LOAD command, or in the *piTempFilesPath* parameter of the **db2Load** API. The default path is a subdirectory of the database directory.

The temporary files path resides on the server machine and is accessed by the DB2 instance exclusively. Therefore, it is imperative that any path name qualification given to the *temp-pathname* parameter reflects the directory structure of the server, not the client, and that the DB2 instance owner has read and write permission on the path.

Note: In an MPP system, the temporary files path should reside on a local disk, not on an NFS mount. If the path is on an NFS mount, there will be significant performance degradation during the load operation.

Attention: The temporary files written to this path must not be tampered with under any circumstances. Doing so will cause the load operation to malfunction, and will place your database in jeopardy.

Related reference:
- "LOAD " on page 132
- "db2Load - Load data into a table" on page 161

Load utility log records

The utility manager produces log records associated with a number of DB2 utilities, including the load utility. The following log records mark the beginning or end of a specific activity during a load operation:
- Load Start. This log record is associated with the beginning of a load operation.
- Load Delete Start. This log record is associated with the beginning of the delete phase in a load operation. The delete phase is started only if there are duplicate primary key values. During the delete phase, each delete operation on a table record, or an index key, is logged.
- Load Delete End. This log record is associated with the end of the delete phase in a load operation. This delete phase will be repeated during the rollforward recovery of a successful load operation.
- Load Pending List. This log record is written when a load transaction commits and is used instead of a normal transaction commit log record.

The following list outlines the log records that the load utility will create depending on the size of the input data:
- Two log records will be created for every table space extent allocated or deleted by the utility in a DMS table space.
- One log record will be created for every chunk of identity values consumed.

- Log records will be created for every data row or index key deleted during the delete phase of a load operation.
- Log records will be created that maintain the integrity of the index tree when performing a load operation with the ALLOW READ ACCESS and INDEXING MODE INCREMENTAL options specified. The number of records logged is considerably less than a fully logged insertion into the index.

Related reference:

- "db2Load - Load data into a table" on page 161
- "LOAD " on page 132

Table locking, table states and table space states

In most cases, the load utility uses table level locking to restrict access to tables. The load utility does not quiesce the table spaces involved in the load operation, and uses table space states only for load operations with the COPY NO option specified. The level of locking depends on whether or not the load operation allows read access. A load operation in ALLOW NO ACCESS mode will use an exclusive lock (Z-lock) on the table for the duration of the load. A load operation in ALLOW READ ACCESS mode acquires and maintains an update lock (U-lock) for the duration of the load operation, and upgrades the lock to an exclusive lock (Z-lock) when data is being committed.

Before a load operation in ALLOW READ ACCESS mode begins, the load utility will wait for all applications that began before the load operation to release locks on the target table. Since locks are not persistent, they are supplemented by table states that will remain even if a load operation is aborted. These states can be checked by using the LOAD QUERY command. By using the LOCK WITH FORCE option, the load utility will force applications holding conflicting locks off the table that it is trying to load into.

Locking Behavior For Load Operations in ALLOW READ ACCESS Mode

At the beginning of a load operation, the load utility acquires an update (U) lock on the table. It holds this lock until the data is being committed. The update lock allows applications with compatible locks to access the table during the load operation. For example, applications that use read only queries will be able to access the table, while applications that try to insert data into the table will be denied. When the load utility acquires the update lock on the table, it will wait for all applications that hold locks on the table prior to the start of the load operation to release them, even if they have compatible locks. This is achieved by temporarily upgrading the update lock to a special exclusive (Z) lock which does not conflict with new table lock requests on the target table as long as the requested locks are compatible with the load operation's update lock. In addition, the load operation upgrades the update lock to an exclusive (Z) lock when the data is being committed, hence there can be some delay in commit time while the load utility waits for applications with conflicting locks to finish.

Note: The load operation can timeout while it waits for the applications to release their locks on the table prior to loading. However, the load operation will not timeout while waiting for the exclusive lock needed to commit the data.

LOCK WITH FORCE Option

The LOCK WITH FORCE option can be used to force off applications holding conflicting locks on the target table so that the load operation can proceed. Applications holding conflicting locks on the system catalog tables will not be forced off by load. If an application is forced off the system by the load utility, it will lose its database connection and an error will be returned (SQL1224N).

For a load operation in ALLOW NO ACCESS mode, all applications holding table locks that exist at the start of the load operation will be forced.

For a load operation in ALLOW READ ACCESS mode applications holding the following locks will be forced:

- Table locks that conflict with a table update lock (for example, import or insert).
- All table locks that exist at the commit phase of the load operation.

When the COPY NO option is specified for a load operation on a recoverable database, all objects in the target table space will be locked in share mode before the table space is placed in backup pending state. This will occur regardless of the access mode. If the LOCK WITH FORCE option is specified, all applications holding locks on objects in the table space that conflict with a share lock will be forced off.

Table States

In addition to locks, the load utility uses table states to control access to tables. A table state can be checked by using the LOAD QUERY command. The states returned by the LOAD QUERY command are as follows:

Normal
No table states affect the table.

Set Integrity Pending
The table has constraints which have not yet been verified. Use the SET INTEGRITY statement to take the table out of Set Integrity Pending state. The load utility places a table in the Set Integrity Pending state when it begins a load operation on a table with constraints.

Load in Progress
There is a load operation in progress on this table.

Load Pending
A load operation has been active on this table but has been aborted before the data could be committed. Issue a LOAD TERMINATE, LOAD RESTART, or LOAD REPLACE command to bring the table out of this state.

Read Access Only
The table data is available for read access queries. Load operations using the ALLOW READ ACCESS option place the table in read access only state.

Unavailable
The table is unavailable. The table can only be dropped or restored from a backup. Rolling forward through a non-recoverable load operation will place a table in the unavailable state.

Not Load Restartable
The table is in a partially loaded state that will not allow a load restart operation. The table will also be in load pending state. Issue a LOAD TERMINATE or a LOAD REPLACE command to bring the table out of the

not load restartable state. A table is placed in not load restartable state when a rollforward operation is performed after a failed load operation that has not been successfully restarted or terminated, or when a restore operation is performed from an online backup that was taken while the table was in load in progress or load pending state. In either case, the information required for a load restart operation is unreliable, and the not load restartable state prevents a load restart operation from taking place.

Type-1 Indexes
The table currently uses type-1 indexes. The indexes can be converted to type-2 using the CONVERT option when using the REORG utility on the indexes.

Unknown
The LOAD QUERY command is unable determine the table state.

A table can be in several states at the same time. For example, if data is loaded into a table with constraints and the ALLOW READ ACCESS option is specified, table state would be:

```
Tablestate:
  Set Integrity Pending
  Load in Progress
  Read Access Only
```

After the load operation but before issuing the SET INTEGRITY statement, the table state would be:

```
Tablestate:
  Set Integrity Pending
  Read Access Only
```

After the SET INTEGRITY statement has been issued the table state would be:

```
Tablestate:
  Normal
```

Table Space States when COPY NO is Specified

If a load operation with the COPY NO option is executed in a recoverable database, the table spaces associated with the load operation are placed in the backup table space state and the load in progress table space state. This takes place at the beginning of the load operation. The load operation can be delayed at this point while locks are acquired on the tables within the table space.

When a table space is in backup pending state, it is still available for read access. The table space can only be taken out of backup pending state by taking a backup of the table space. Even if the load operation is aborted, the table space will remain in backup pending state because the table space state is changed at the beginning of the load operation, and cannot be rolled back if it fails. The load in progress table space state prevents online backups of a load operation with the COPY NO option specified while data is being loaded. The load in progress state is removed when the load operation is completed or aborts.

During a rollforward operation through a LOAD command with the COPY NO option specified, the associated table spaces are placed in restore pending state. To remove the table spaces from restore pending state, a restore operation must be performed. A rollforward operation will only place a table space in the restore pending state if the load operation completed successfully.

Related concepts:

- "Pending states after a load operation" on page 206

Character set and national language support

The DB2 data movement utilities offer the following National Language Support (NLS):

- The import and the export utilities provide automatic code page conversion from a client code page to the server code page.
- For the load utility, data can be converted from any code page to the server code page by using the `codepage` modifier with DEL and ASC files.
- For all utilities, IXF data is automatically converted from its original code page (as stored in the IXF file) to the server code page.

Unequal code page situations, involving expansion or contraction of the character data, can sometimes occur. For example, Japanese or Traditional-Chinese Extended UNIX Code (EUC) and double-byte character sets (DBCS) might encode different lengths for the same character. Normally, comparison of input data length to target column length is performed before reading in any data. If the input length is greater than the target length, NULLs are inserted into that column if it is nullable. Otherwise, the request is rejected. If the `nochecklengths` modifier is specified, no initial comparison is performed, and an attempt is made to load the data. If the data is too long after translation is complete, the row is rejected. Otherwise, the data is loaded.

Related reference:

- "LOAD " on page 132

Pending states after a load operation

The load utility uses *table* states to preserve database consistency during a load operation. These states can be checked by using the LOAD QUERY command.

The load and build phases of the load process place the target table in the load in progress table state. The load utility also places table spaces in the load in progress state when the COPY NO option is specified on a recoverable database. The table spaces remain in this state for the duration of the load operation and are returned to normal state if the transaction is committed or rolled back.

If the NO ACCESS option has been specified, the table cannot be accessed while the load is in progress. If the ALLOW READ ACCESS option has been specified, the data in the table that existed prior to the invocation of the load command will be available in read only mode during the load operation. If the ALLOW READ ACCESS option is specified and the load operation fails, the data that existed in the table prior to the load operation will continue to be available in read only mode after the failure.

To remove the load in progress table state (if the load operation has failed, or was interrupted), do one of the following:

- Restart the load operation. First, address the cause of the failure; for example, if the load utility ran out of disk space, add containers to the table space before attempting a load restart operation.
- Terminate the load operation.

- Invoke a LOAD REPLACE operation against the same table on which a load operation has failed.
- Recover table spaces for the loading table by using the RESTORE DATABASE command with the most recent table space or database backup, and then carry out further recovery actions.

During a load operation, table spaces are placed in backup pending after the first commit, and:

- The database is recoverable (database configuration parameter *logarchmeth1* or *logarchmeth2* is not set to OFF) and
- The load option COPY YES is not specified, and
- The load option NONRECOVERABLE is not specified.

The fourth possible state associated with the load process (Set Integrity Pending state) pertains to referential and check constraints, generated column constraints, materialized query computation, or staging table propagation. For example, if an existing table is a parent table containing a primary key referenced by a foreign key in a dependent table, replacing data in the parent table places both tables (not the table space) in Set Integrity Pending state. To validate a table for referential integrity and check constraints, issue the SET INTEGRITY statement after the load process completes, if the table has been left in Set Integrity Pending state.

Related concepts:

- "Checking for integrity violations following a load operation" on page 121
- "Table locking, table states and table space states" on page 203

Related reference:

- "LIST TABLESPACES command" in *Command Reference*

Optimizing load performance

The performance of the load utility depends on the nature and the quantity of the data, the number of indexes, and the load options specified.

Unique indexes reduce load performance if duplicates are encountered. In most cases, it is still more efficient to create indexes during the load operation than to invoke the CREATE INDEX statement for each index after the load operation completes (see Figure 5 on page 208).

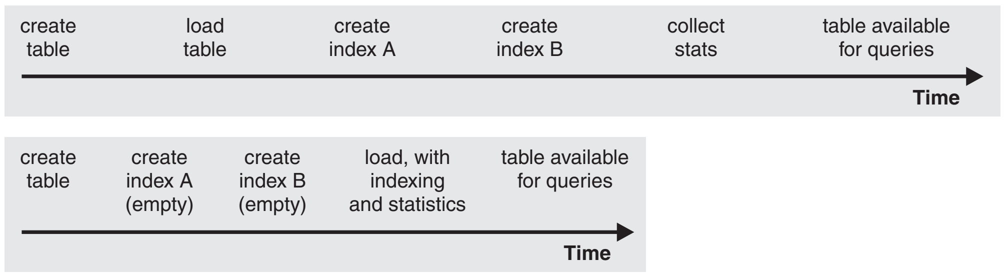


Figure 5. Increasing load performance through concurrent indexing and statistics collection. Tables are normally built in three steps: data loading, index building, and statistics collection. This causes multiple data I/O during the load operation, during index creation (there can be several indexes for each table), and during statistics collection (which causes I/O on the table data and on all of the indexes). A much faster alternative is to let the load utility complete all of these tasks in one pass through the data.

When tuning index creation performance, the amount of memory dedicated to the sorting of index keys during a load operation is controlled by the *sortheap* database configuration parameter. For example, to direct the load utility to use 4000 pages of main memory per index for key sorting, set the *sortheap* database configuration parameter to be 4000 pages, disconnect all applications from the database, and then issue the LOAD command. If an index is so large that it cannot be sorted in memory, a sort spill occurs. That is, the data is divided among several ″sort runs″ and stored in a temporary table space that is merged later. If there is no way to avoid a sort spill by increasing the size of the *sortheap* parameter, it is important that the buffer pool for temporary table spaces be large enough to minimize the amount of disk I/O that spilling causes. Furthermore, to achieve I/O parallelism during the merging of sort runs, it is recommended that temporary table spaces be declared with multiple containers, each residing on a different disk device. If there is more than one index defined on a table, memory consumption increases proportionally because the load operation keeps all keys in memory.

Sorting during index rebuild uses up to SORTHEAP pages. If more is required, TEMP buffer pool is used and (eventually) spilled to disk. If load spills, and thus decreases performance, it might be advisable to run LOAD with INDEXING MODE DEFERRED and recreate the index later. CREATE INDEX creates one index at a time, reducing memory usage while scanning the table multiple times to collect keys.

Load operations with the ALLOW READ ACCESS and INDEXING MODE REBUILD options allow you to specify the USE <table space> option for storing a shadow index. While the index still has to be copied to the target table space before becoming visible, this option minimizes use of the target table space while a load operation is in progress.

For Index Rebuild, load uses a single table scanner, which also does the sorting, to pick up existing keys and create indexes. Multiple table scanners are used with Index Manager code (IXM), which builds the indexes outside of the load operation.

The advantage of building the indexes with a CREATE INDEX statement instead of a load operation is that the CREATE INDEX statement can use multiple processes (also known as threads) to sort keys if INTRA PARALLEL is on. The actual building of the index is not executed in parallel.

Use of the SET INTEGRITY statement might lengthen the total time needed for a table to become usable again. If all the load operations are performed in INSERT mode, the SET INTEGRITY statement checks the table for integrity violations incrementally (by checking only the appended portion of the table). If a table cannot be checked for integrity violations incrementally, the entire table is checked, and it might be some time before the table is usable again.

Similarly, if a load operation is performed on the underlying tables of a materialized query table, use of the REFRESH TABLE statement might lengthen the time needed to make both the underlying tables and the materialized query table fully usable again. If several sequential load operations are performed in INSERT mode into the underlying tables of a REFRESH IMMEDIATE materialized query table, the SET INTEGRITY statement incrementally refreshes the materialized query table in most cases. If the system determines that a full refresh is required, the materialized query table definition query is recomputed, and it might be some time before the table is usable again.

The load utility performs equally well in INSERT mode and in REPLACE mode. However, if indexing mode is REBUILD, REPLACE mode will perform better than INSERT mode because there is no need to scan existing data.

The utility attempts to deliver the best performance possible by determining optimal values for DISK_PARALLELISM, CPU_PARALLELISM, and DATA BUFFER, if these parameters have not be specified by the user. Optimization is done based on the size and the free space available in the utility heap. Consider using the autonomic DISK_PARALLELISM and CPU_PARALLELISM settings before attempting to tune these parameters for your particular needs.

Following is information about the performance implications of various options available through the load utility:

ANYORDER
: Specify this file type modifier to suspend the preservation of order in the data being loaded, and improve performance. If the data to be loaded is presorted, `anyorder` might corrupt the presorted order, and the benefits of presorting is lost for subsequent queries.

BINARY NUMERICS and PACKED DECIMAL
: Use these file type modifiers to improve performance when loading positional numeric ASC data into fixed-length records.

COPY YES or NO
: Use this parameter to specify whether a copy of the input data is to be made during a load operation. COPY YES reduces load performance, because all of the loading data is copied during the load operation (forward recovery must be enabled); the increased I/O activity might increase the load time on an I/O-bound system. Specifying multiple devices or directories (on different disks) can offset some of the performance penalty resulting from this operation. COPY NO might reduce overall performance, because if forward recovery is enabled, the table space is placed in backup pending state, and the database, or selected table spaces, must be backed up before the table can be accessed.

CPU_PARALLELISM
: Use this parameter to exploit intra-partition parallelism (if this is part of your machine's capability), and significantly improve load performance. The parameter specifies the number of processes or threads used by the load utility to parse, convert, and format data records. The maximum number allowed is 30. If there is insufficient memory to support the

specified value, the utility adjusts the value. If this parameter is not specified, the load utility selects a default value that is based on the number of CPUs on the system.

Record order in the source data is preserved (see Figure 6) regardless of the value of this parameter.

If tables include either LOB or LONG VARCHAR data, CPU_PARALLELISM is set to one. Parallelism is not supported in this case.

Although use of this parameter is not restricted to symmetric multiprocessor (SMP) hardware, you might not obtain any discernible performance benefit from using it in non-SMP environments.

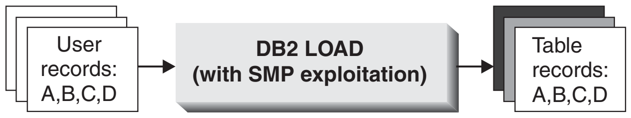


Figure 6. Record Order in the Source Data is Preserved When Intra-partition Parallelism is Exploited During a Load Operation

DATA BUFFER
The DATA BUFFER parameter specifies the total amount of memory allocated to the load utility as a buffer. It is recommended that this buffer be several *extents* in size. An extent is the unit of movement for data within DB2, and the extent size can be one or more 4KB pages. The DATA BUFFER parameter is useful when working with large objects (LOBs); it reduces I/O waiting time. The data buffer is allocated from the utility heap. Depending on the amount of storage available on your system, you should consider allocating more memory for use by the DB2 utilities. The database configuration parameter *util_heap_sz* can be modified accordingly. The default value for the Utility Heap Size configuration parameter is 5 000 4KB pages. Because load is only one of several utilities that use memory from the utility heap, it is recommended that no more than fifty percent of the pages defined by this parameter be available for the load utility, and that the utility heap be defined large enough.

DISK_PARALLELISM
The DISK_PARALLELISM parameter specifies the number of processes or threads used by the load utility to write data records to disk. Use this parameter to exploit available containers when loading data, and significantly improve load performance. The maximum number allowed is the greater of four times the CPU_PARALLELISM value (actually used by the load utility), or 50. By default, DISK_PARALLELISM is equal to the sum of the table space containers on all table spaces containing objects for the table being loaded, except where this value exceeds the maximum number allowed.

FASTPARSE
Use the `fastparse` file type modifier to reduce the data checking that is performed on user-supplied column values, and enhance performance. This option should only be used when the data being loaded is known to be valid. It can improve performance by about 10 or 20 percent.

NONRECOVERABLE
Use this parameter if you do not need to be able to recover load transactions against a table. Load performance is enhanced, because no

additional activity beyond the movement of data into the table is required, and the load operation completes without leaving the table spaces in backup pending state.

Note: When these load transactions are encountered during subsequent restore and rollforward recovery operations, the table is not updated, and is marked "invalid". Further actions against this table are ignored. After the rollforward operation is complete, the table can either be dropped or a LOAD TERMINATE command can be issued to bring it back online.

NOROWWARNINGS
Use the `norowwarnings` file type modifier to suppress the recording of warnings about rejected rows, and enhance performance, if you anticipate a large number of warnings.

ALLOW READ ACCESS
This option allows you to query a table while a load operation is in progress. You can only view data that existed in the table prior to the load operation. If the INDEXING MODE INCREMENTAL option is also specified, and the load operation fails, the subsequent load terminate operation might have to correct inconsistencies in the index. This requires an index scan which involves considerable I/O. If the ALLOW READ ACCESS option is also specified for the load terminate operation, the buffer pool is used for I/O.

SAVECOUNT
Use this parameter to set an interval for the establishment of consistency points during a load operation. The synchronization of activities performed to establish a consistency point takes time. If done too frequently, there is a noticeable reduction in load performance. If a very large number of rows is to be loaded, it is recommended that a large SAVECOUNT value be specified (for example, a value of ten million in the case of a load operation involving 100 million records).

A LOAD RESTART operation automatically continues from the last consistency point.

STATISTICS USE PROFILE
Collect statistics specified in table statistics profile. Use this parameter to collect data distribution and index statistics more efficiently than through invocation of the runstats utility following completion of the load operation, even though performance of the load operation itself decreases (particularly when DETAILED INDEXES ALL is specified).

For optimal performance, applications require the best data distribution and index statistics possible. Once the statistics are updated, applications can use new access paths to the table data based on the latest statistics. New access paths to a table can be created by rebinding the application packages using the DB2 BIND command. The table statistics profile is created by running the RUNSTATS command with the SET PROFILE options.

When loading data into large tables, it is recommended that a larger value for the *stat_heap_sz* (Statistics Heap Size) database configuration parameter be specified.

USE <tablespaceName>
This parameter allows an index to be rebuilt in a system temporary table space and copied back to the index table space during the index copy

phase of a load operation. When a load operation in ALLOW READ ACCESS mode fully rebuilds the indexes, the new indexes are built as a *shadow*. The original indexes are replaced by the new indexes at the end of the load operation.

By default, the *shadow* index is built in the same table space as the original index. This might cause resource problems as both the original and the *shadow* index reside in the same table space simultaneously. If the *shadow* index is built in the same table space as the original index, the original index is instantaneously replaced by the *shadow*. However, if the *shadow* index is built in a system temporary table space, the load operation requires an index copy phase which copies the index from a system temporary table space to the index table space. There is considerable I/O involved in the copy. If either of the table spaces is a DMS table space, the I/O on the system temporary table space might not be sequential. The values specified by the DISK_PARALLELISM option are respected during the index copy phase.

WARNINGCOUNT
Use this parameter to specify the number of warnings that can be returned by the utility before a load operation is forced to terminate. If you are expecting only a few warnings or no warnings, set the WARNINGCOUNT parameter to approximately the number you are expecting, or to twenty if you are expecting no warnings. The load operation stops after the WARNINGCOUNT number is reached. This gives you the opportunity to correct data (or to drop and then recreate the table being loaded) before attempting to complete the load operation. Although not having a direct effect on the performance of the load operation, the establishment of a WARNINGCOUNT threshold prevents you from having to wait until the entire load operation completes before determining that there is a problem.

Related concepts:

- "DB2 registry and environment variables" in *Performance Guide*
- "Multidimensional clustering considerations " on page 125

Related reference:

- "SET INTEGRITY statement" in *SQL Reference, Volume 2*
- "BIND command" in *Command Reference*
- "UPDATE DATABASE CONFIGURATION command" in *Command Reference*
- "stat_heap_sz - Statistics heap size configuration parameter" in *Performance Guide*
- "util_heap_sz - Utility heap size configuration parameter" in *Performance Guide*

Load - CLP examples

Example 1

TABLE1 has 5 columns:

- COL1 VARCHAR 20 NOT NULL WITH DEFAULT
- COL2 SMALLINT
- COL3 CHAR 4
- COL4 CHAR 2 NOT NULL WITH DEFAULT
- COL5 CHAR 2 NOT NULL

ASCFILE1 has 6 elements:

- ELE1 positions 01 to 20
- ELE2 positions 21 to 22
- ELE3 positions 23 to 23
- ELE4 positions 24 to 27
- ELE5 positions 28 to 31
- ELE6 positions 32 to 32
- ELE7 positions 33 to 40

Data Records:

```
1...5...10...15...20...25...30...35...40
Test data 1         XXN 123abcdN
Test data 2 and 3   QQY    XXN
Test data 4,5 and 6 WWN6789    Y
```

The following command loads the table from the file:

```
db2 load from ascfile1 of asc modified by striptblanks reclen=40
   method L (1 20, 21 22, 24 27, 28 31)
   null indicators (0,0,23,32)
   insert into table1 (col1, col5, col2, col3)
```

Notes:

1. The specification of `striptblanks` in the MODIFIED BY parameter forces the truncation of blanks in VARCHAR columns (COL1, for example, which is 11, 17 and 19 bytes long, in rows 1, 2 and 3, respectively).
2. The specification of `reclen=40` in the MODIFIED BY parameter indicates that there is no new-line character at the end of each input record, and that each record is 40 bytes long. The last 8 bytes are not used to load the table.
3. Since COL4 is not provided in the input file, it will be inserted into TABLE1 with its default value (it is defined NOT NULL WITH DEFAULT).
4. Positions 23 and 32 are used to indicate whether COL2 and COL3 of TABLE1 will be loaded NULL for a given row. If there is a `Y` in the column's null indicator position for a given record, the column will be NULL. If there is an `N`, the data values in the column's data positions of the input record (as defined in L(........)) are used as the source of column data for the row. In this example, neither column in row 1 is NULL; COL2 in row 2 is NULL; and COL3 in row 3 is NULL.
5. In this example, the NULL INDICATORS for COL1 and COL5 are specified as 0 (zero), indicating that the data is not nullable.
6. The NULL INDICATOR for a given column can be anywhere in the input record, but the position must be specified, and the `Y` or `N` values must be supplied.

Example 2 (Using Dump Files)

Table FRIENDS is defined as:

```
table friends "( c1 INT NOT NULL, c2 INT, c3 CHAR(8) )"
```

If an attempt is made to load the following data records into this table,

```
23, 24, bobby
, 45, john
4,, mary
```

the second row is rejected because the first INT is NULL, and the column definition specifies NOT NULL. Columns which contain initial characters that are not consistent with the DEL format will generate an error, and the record will be rejected. Such records can be written to a dump file.

DEL data appearing in a column outside of character delimiters is ignored, but does generate a warning. For example:

```
22,34,"bob"
24,55,"sam" sdf
```

The utility will load ″sam″ in the third column of the table, and the characters ″sdf″ will be flagged in a warning. The record is not rejected. Another example:

```
22 3, 34,"bob"
```

The utility will load `22,34,"bob"`, and generate a warning that some data in column one following the `22` was ignored. The record is not rejected.

Example 3 (Loading a Table with an Identity Column)

TABLE1 has 4 columns:

- C1 VARCHAR(30)
- C2 INT GENERATED BY DEFAULT AS IDENTITY
- C3 DECIMAL(7,2)
- C4 CHAR(1)

TABLE2 is the same as TABLE1, except that C2 is a GENERATED ALWAYS identity column.

Data records in DATAFILE1 (DEL format):

```
"Liszt"
"Hummel",,187.43, H
"Grieg",100, 66.34, G
"Satie",101, 818.23, I
```

Data records in DATAFILE2 (DEL format):

```
"Liszt", 74.49, A
"Hummel", 0.01, H
"Grieg", 66.34, G
"Satie", 818.23, I
```

Notes:

1. The following command generates identity values for rows 1 and 2, since no identity values are supplied in DATAFILE1 for those rows. Rows 3 and 4, however, are assigned the user-supplied identity values of 100 and 101, respectively.
   ```
   db2 load from datafile1.del of del replace into table1
   ```
2. To load DATAFILE1 into TABLE1 so that identity values are generated for all rows, issue one of the following commands:
   ```
   db2 load from datafile1.del of del method P(1, 3, 4)
      replace into table1 (c1, c3, c4)
   db2load from datafile1.del of del modified by identityignore
      replace into table1
   ```
3. To load DATAFILE2 into TABLE1 so that identity values are generated for each row, issue one of the following commands:

```
db2 load from datafile2.del of del replace into table1 (c1, c3, c4)
db2 load from datafile2.del of del modified by identitymissing
   replace into table1
```

4. To load DATAFILE1 into TABLE2 so that the identity values of 100 and 101 are assigned to rows 3 and 4, issue the following command:

   ```
   db2 load from datafile1.del of del modified by identityoverride
      replace into table2
   ```

   In this case, rows 1 and 2 will be rejected, because the utility has been instructed to override system-generated identity values in favor of user-supplied values. If user-supplied values are not present, however, the row must be rejected, because identity columns are implicitly not NULL.
5. If DATAFILE1 is loaded into TABLE2 without using any of the identity-related file type modifiers, rows 1 and 2 will be loaded, but rows 3 and 4 will be rejected, because they supply their own non-NULL values, and the identity column is GENERATED ALWAYS.

Example 4 (Loading from CURSOR)

MY.TABLE1 has 3 columns:
- ONE INT
- TWO CHAR(10)
- THREE DATE

MY.TABLE2 has 3 columns:
- ONE INT
- TWO CHAR(10)
- THREE DATE

Cursor MYCURSOR is defined as follows:

```
declare mycursor cursor for select * from my.table1
```

The following command loads all the data from MY.TABLE1 into MY.TABLE2:

```
load from mycursor of cursor method P(1,2,3) insert into
  my.table2(one,two,three)
```

Notes:
1. Only one cursor name can be specified in a single LOAD command. That is, `load  from mycurs1, mycurs2 of cursor...` is not allowed.
2. `P` and `N` are the only valid METHOD values for loading from a cursor.
3. In this example, METHOD P and the insert column list `(one,two,three)` could have been omitted since they represent default values.
4. MY.TABLE1 can be a table, view, alias, or nickname.

Related concepts:
- "Load overview" on page 102

Related reference:
- "Examples of loading data in a partitioned database environment" on page 234
- "LOAD " on page 132
- "LOAD QUERY " on page 158

Chapter 4. Loading data in a partitioned database environment

This chapter describes loading data in a partitioned database environment.

The following topics are covered:

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.

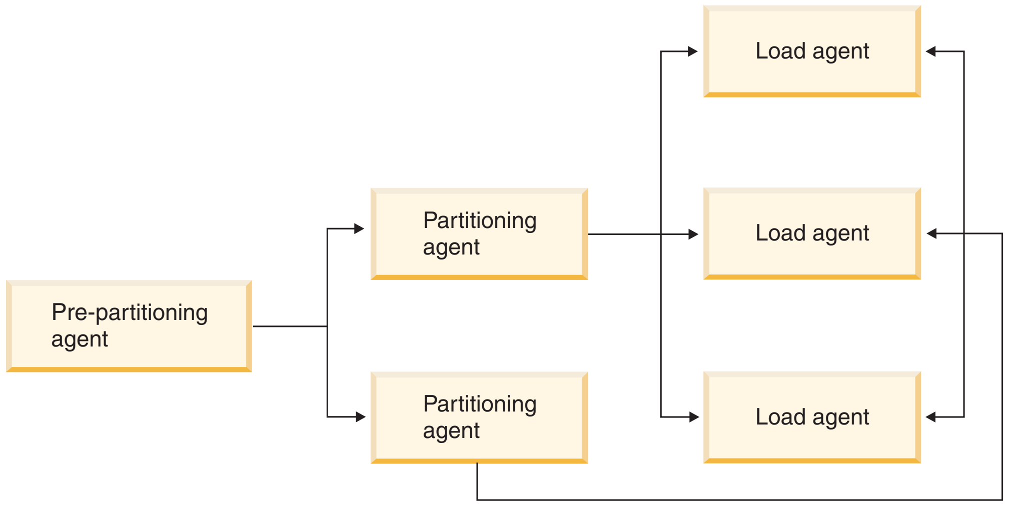


Figure 7. Partitioned Database Load Overview. The source data is read by the pre-partitioning agent, approximately half of the data is sent to each of two partitioning agents which distribute the data and send it to one of three database partitions. The load agent at each database partition loads the data.

Related concepts:

- "Data organization schemes" in *Administration Guide: Planning*
- "Load overview" on page 102
- "Loading data in a partitioned database environment - hints and tips" on page 237
- "Monitoring a load operation in a partitioned database environment using the LOAD QUERY command" on page 225
- "Restarting or terminating a load operation in a partitioned database environment" on page 227

Related tasks:

- "Loading data" on page 110
- "Loading data in a partitioned database environment" on page 219

Related reference:

- "Load configuration options for partitioned database environments" on page 229

Loading data in a partitioned database environment

Prerequisites:

Before loading a table in a multi-partition database:

1. Ensure that the *svcename* database manager configuration parameter and the **DB2COMM** profile registry variable are set correctly. This is important because the load utility uses TCP/IP to transfer data from the pre-partitioning agent to the partitioning agents, and from the partitioning agents to the loading database partitions.
2. Before invoking the load utility, you must be connected to (or be able to implicitly connect to) the database into which the data will be loaded. Since the load utility will issue a COMMIT statement, you should complete all transactions and release any locks by issuing either a COMMIT or a ROLLBACK statement before beginning the load operation. If the

PARTITION_AND_LOAD, PARTITION_ONLY, or ANALYZE mode is being used, the data file that is being loaded must reside on this database partition unless:

   a. the CLIENT option has been specified, in which case the data must reside on the client machine;
   b. the input source type is CURSOR, in which case there is no input file.

3. Run the Design Advisor to determine the best database partition for each table. For more information, see The Design Advisor.

Restrictions:

The following restrictions apply when using the load utility to load data in a multi-partition database:

- The location of the input files to the load operation cannot be a tape device.
- The ROWCOUNT option is not supported unless the ANALYZE mode is being used.
- If the target table has an identity column that is needed for distributing and the *identityoverride* modifier is not specified, or if you are using multiple database partitions to distribute and then load the data, the use of a SAVECOUNT greater than zero on the LOAD command is not supported.
- If an identity column forms part of the distribution key, only PARTITION_AND_LOAD mode is supported.
- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART modes cannot be used with the CLIENT option of the LOAD command.
- The LOAD_ONLY_VERIFY_PART mode cannot be used with the CURSOR input source type.
- The distribution error isolation modes LOAD_ERRS_ONLY and SETUP_AND_LOAD_ERRS cannot be used with the ALLOW READ ACCESS and COPY YES options of the LOAD command.
- Multiple load operations can load data into the same table concurrently if the database partitions specified by the OUTPUT_DBPARTNUMS and PARTITIONING_DBPARTNUMS options do not overlap. For example, if a table is defined on database partitions 0 through 3, one load operation can load data into database partitions 0 and 1 while a second load operation can load data into database partitions 2 and 3.
- Only Non-delimited ASCII (ASC) and Delimited ASCII (DEL) files can be distributed across tables spanning multiple database partitions. PC/IXF files cannot be distributed. To load a PC/IXF file into a table spanning multiple database partitions, you can first load it into a table residing on a single database partition by setting the environment variable DB2_PARTITIONEDLOAD_DEFAULT=NO and using the LOAD_ONLY_VERIFY_PART mode. Then you can perform a load operation using the CURSOR file type to move the data into a table that is distributed over multiple database partitions. You can also load a PC/IXF file into a table that is distributed over multiple database partitions using the load operation in the LOAD_ONLY_VERIFY_PART mode.

Procedure:

The following examples illustrate how to use the LOAD command to initiate various types of load operations. The database used in the following examples has five database partitions: 0, 1, 2, 3 and 4. Each database partition has a local directory `/db2/data/`. Two tables, TABLE1 and TABLE2, are defined on database partitions 0, 1, 3 and 4. When loading from a client, the user has access to a remote client that is not one of the database partitions.

Loading from a server partition

Distribute and load example

In this scenario you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To load the data from `load.del` into all of the database partitions where TABLE1 is defined, issue the following command: `LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1`

Note: In this example, default values are used for all of the configuration parameters for partitioned database environments: The MODE parameter default to PARTITION_AND_LOAD, the OUTPUT_DBPARTNUMS options default to all database partitions on which TABLE1 is defined, and the PARTITIONING_DBPARTNUMS defaults to the set of database partitions selected according to the LOAD command rules for choosing database partitions when none are specified.

To perform a load operation where data is distributed over database partitions 3 and 4, issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                     PARTITIONED DB CONFIG PARTITIONING_DBPARTNUMS (3,4)
```

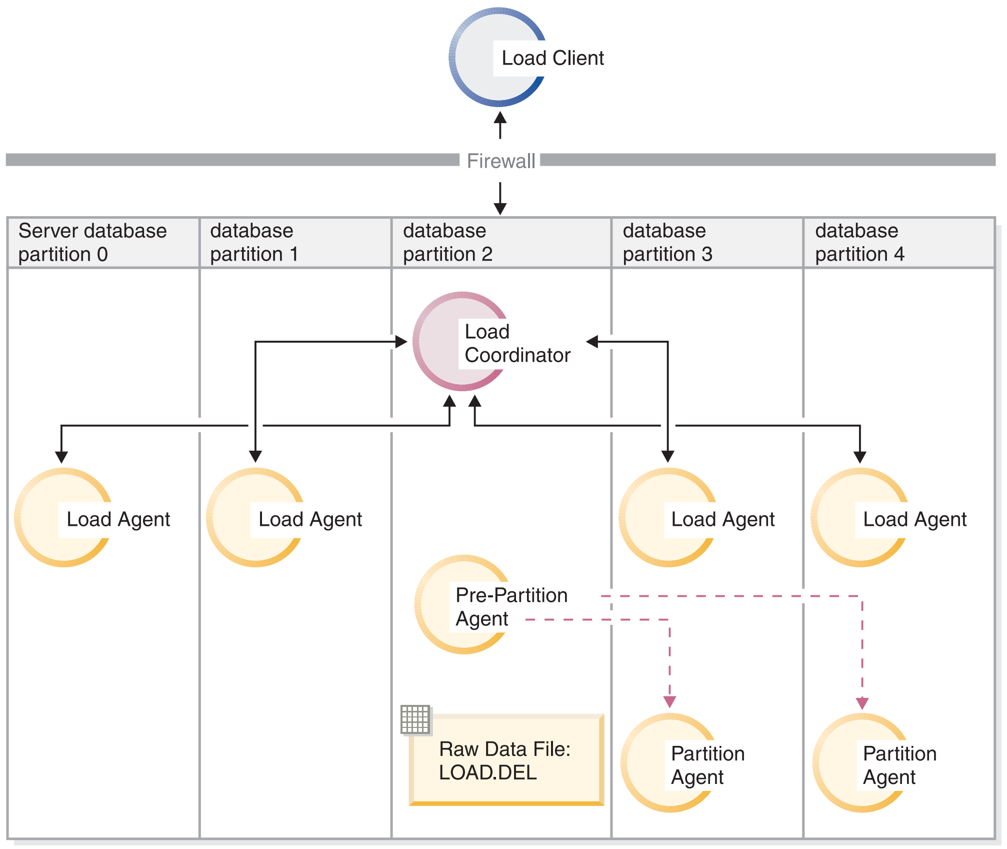


Figure 8. . This diagram illustrates the behavior resulting when the previous command is issued. Data is loaded into database partitions 3 and 4.

Distribute only example

In this scenario you are connected to a database partition that might or might not be a database partition where TABLE1 is defined. The data file `load.del` resides in the current working directory of this database partition. To distribute (but not load) `load.del` to all the database partitions on which TABLE1 is defined, using database partitions 3 and 4 issue the following command:

```
LOAD FROM LOAD.DEL of DEL REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                             PART_FILE_LOCATION /db2/data
                                             PARTITIONING_DBPARTNUMS (3,4)
```

This results in a file `load.del.xxx` being stored in the `/db2/data` directory on each database partition, where `xxx` is a three-digit representation of the database partition number.

To distribute the `load.del` file to database partitions 1 and 3, using only 1 partitioning agent running on database partition 0 (which is the default for PARTITIONING_DBPARTNUMS), issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE PARTITION_ONLY
                                             PART_FILE_LOCATION /db2/data
                                             OUTPUT_DBPARTNUMS (1,3)
```

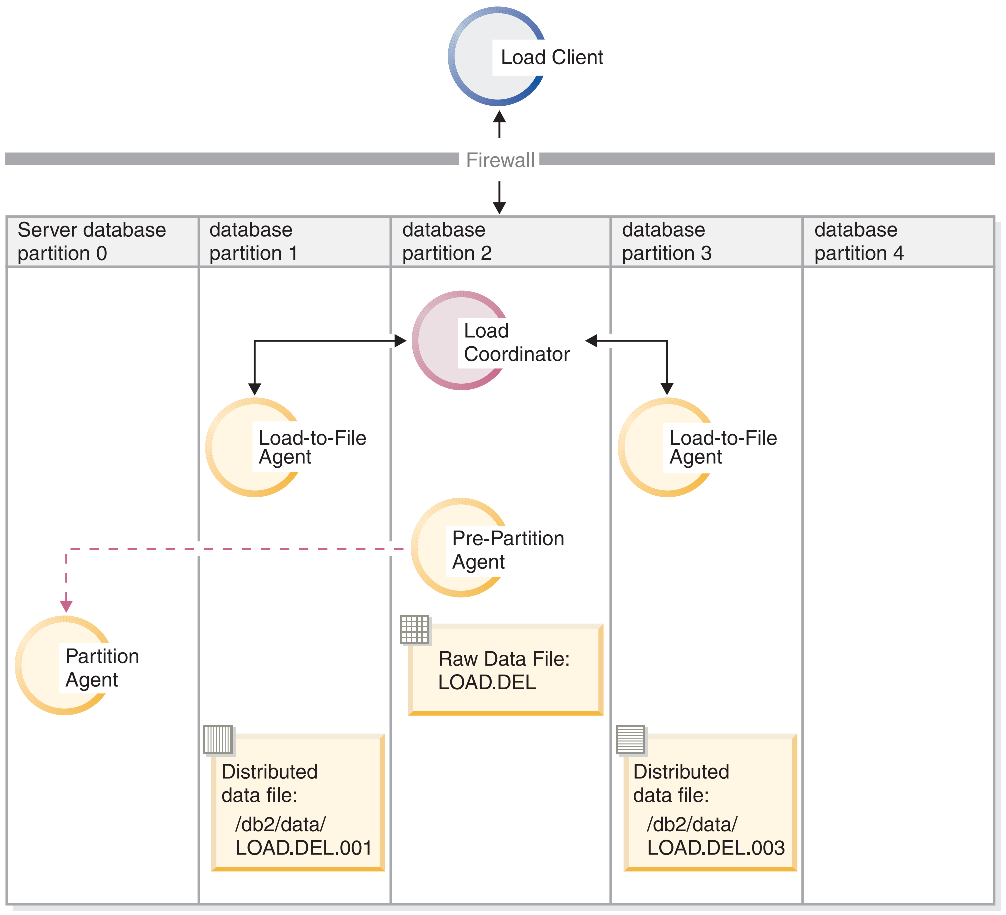


Figure 9. . This diagram illustrates the behavior that results when the previous command is issued. Data is loaded into database partitions 1 and 3, using 1 partitioning agent running on database partition 0.

Load only example

If you have already performed a load operation in the PARTITION_ONLY mode and want to load the partitioned files in the `/db2/data` directory of each loading database partition to all the database partitions on which TABLE1 is defined, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                     PARTITIONED DB CONFIG MODE LOAD_ONLY
                                         PART_FILE_LOCATION /db2/data
```

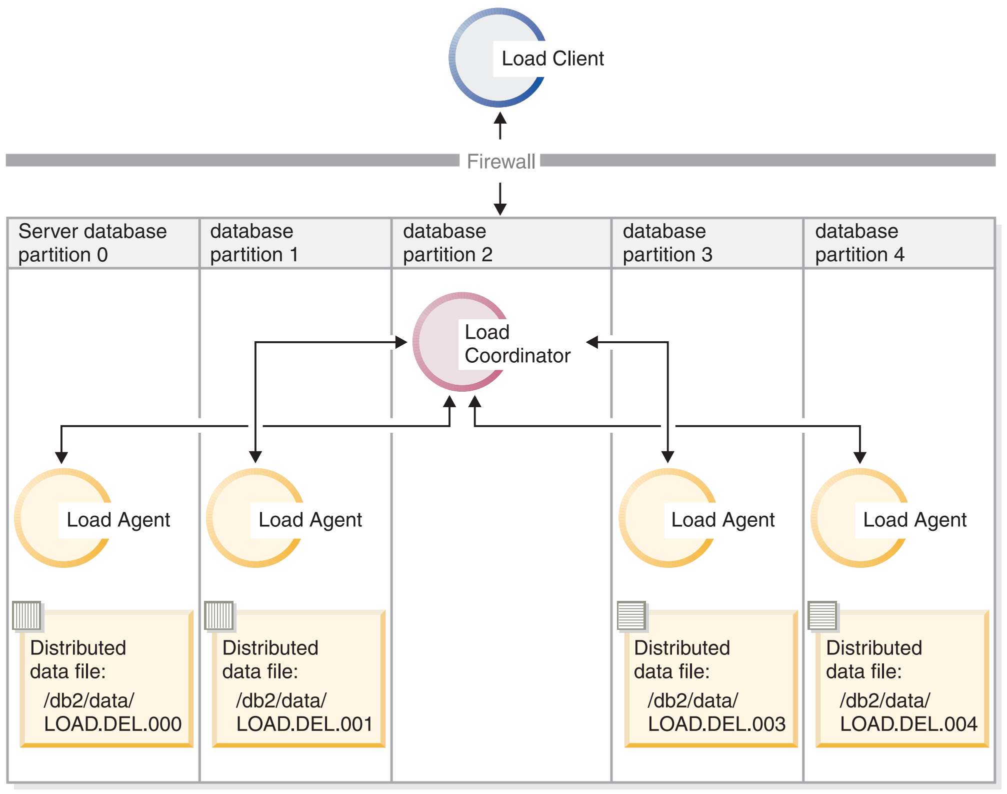

Load Client
Firewall
Server database partition 0
database partition 1
database partition 2
database partition 3
database partition 4
Load Coordinator
Load Agent
Load Agent
Load Agent
Load Agent
Distributed data file: /db2/data/ LOAD.DEL.000
Distributed data file: /db2/data/ LOAD.DEL.001
Distributed data file: /db2/data/ LOAD.DEL.003
Distributed data file: /db2/data/ LOAD.DEL.004

Figure 10. . This diagram illustrates the behavior resulting when the previous command is issued. Distributed data is loaded to all database partitions where TABLE1 is defined.

To load into database partition 4 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY
                                             PART_FILE_LOCATION /db2/data
                                             OUTPUT_DBPARTNUMS (4)
```

Loading pre-distributed files without distribution map headers

The LOAD command can be used to load data files without distribution headers directly into several database partitions. If the data files exist in the `/db2/data` directory on each database partition where TABLE1 is defined and have the name `load.del.xxx`, where `xxx` is the database partition number, the files can be loaded by issuing the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
                       REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
                                             PART_FILE_LOCATION /db2/data
```

To load the data into database partition 1 only, issue the following command:

```
LOAD FROM LOAD.DEL OF DEL modified by dumpfile=rejected.rows
                       REPLACE INTO TABLE1
                       PARTITIONED DB CONFIG MODE LOAD_ONLY_VERIFY_PART
                                             PART_FILE_LOCATION /db2/data
                                             OUTPUT_DBPARTNUMS (1)
```

Note: Rows that do not belong on the database partition from which they were loaded are rejected and put into the dumpfile, if one has been specified.

Loading from a remote client to a multi-partition database

To load data into a multi-partition database from a file that is on a remote client, you must specify the CLIENT option of the LOAD command to indicate that the data file is not on a server partition. For example:

```
LOAD CLIENT FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

Note: You cannot use the LOAD_ONLY or LOAD_ONLY_VERIFY_PART modes with the CLIENT option.

Loading from a cursor

As in a single-partition database, you can load from a cursor into a multi-partition database. In this example, for the PARTITION_ONLY and LOAD_ONLY modes, the PART_FILE_LOCATION option must specify a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created or loaded on each output database partition. Multiple files can be created with the specified base name if there are LOB columns in the target table.

To distribute all the rows in the answer set of the statement `SELECT * FROM  TABLE1` to a file on each database partition named `/db2/data/select.out.xxx` (where xxx is the database partition number), for future loading into TABLE2, issue the following commands:

```
DECLARE C1 CURSOR FOR SELECT * FROM TABLE1

LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED DB CONFIG MODE PARTITION_ONLY
                       PART_FILE_LOCATION /db2/data/select.out
```

The data files produced by the above operation can then be loaded by issuing the following LOAD command:

```
LOAD FROM C1 OF CURSOR REPLACE INTO TABLE2
                       PARTITIONED CB CONFIG MODE LOAD_ONLY
                       PART_FILE_LOCATION /db2/data/select.out
```

Related concepts:

- “The Design Advisor” in *Performance Guide*
- “Moving data using the CURSOR file type” on page 267

Related reference:

- “db2Load - Load data into a table” on page 161

Monitoring a load operation in a partitioned database environment using the LOAD QUERY command

Message files produced during a multi-partition database load operation

During a load operation, message files are created by some of the load processes on the database partitions where they are being executed. These files store all information, warning and error messages produced during the execution of the load operation. The load processes that produce message files that can be viewed by the user are the load agent, pre-partitioning agent and partitioning agent.

You can connect to individual database partitions during a load operation and issue the LOAD QUERY command against the target table. When issued from the

CLP, this command displays the contents of all the message files that currently reside on that database partition for the table that is specified in the LOAD QUERY command.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. You are connected to database partition 0 and issue the following LOAD command:

```
load from load.del of del replace into table1 partitioned db config
    partitioning_dbpartnums (1)
```

This command initiates a load operation that includes load agents running on database partitions 0, 1, 2 and 3; a partitioning agent running on database partition 1; and a pre-partitioning agent running on database partition 0.

Database partition 0 contains one message file for the pre-partitioning agent and one for the load agent on that database partition. To view the contents of these files at the same time, start a new session and issue the following commands from the CLP:

```
set client connect_node 0
connect to wsdb
load query table table1
```

Database partition 1 contains one file for the load agent and one for the partitioning agent. To view the contents of these files, start a new session and issue the following commands from the CLP:

```
set client connect_node 1
connect to wsdb
load query table table1
```

Note: The messages generated by the STATUS_INTERVAL load configuration option appear in the pre-partitioning agent message file. To view these message during a load operation, you must connect to the coordinator partition and issue the LOAD QUERY command.

Saving the contents of message files

If a load operation is initiated through the **db2Load** API, the messages option (piLocalMsgFileName) must be specified and the message files are brought from the server to the client and stored for you to view.

For multi-partition database load operations initiated from the CLP, the message files are not displayed to the console or retained. To save or view the contents of these files after a multi-partition database load is complete, the MESSAGES option of the LOAD command must be specified. If this option is used, once the load operation is complete the message files on each database partition are transferred to the client machine and stored in files using the base name indicated by the MESSAGES option. For multi-partition database load operations, the name of the file corresponding to the load process that produced it is listed below:

Process Type	**File Name**
Load Agent	<message-file-name>.LOAD.<dbpartition-number>
Partitioning Agent	<message-file-name>.PART.<dbpartition-number>

Process Type	File Name
Pre-partitioning Agent	<message-file-name>.PREP.<dbpartition-number>

For example, if the MESSAGES option specifies `/wsdb/messages/load`, the load agent message file for database partition 2 is `/wsdb/messages/load.LOAD.002`.

Note: It is strongly recommended that the MESSAGES option be used for multi-partition database load operations initiated from the CLP.

Related reference:
- "db2LoadQuery - Get the status of a load operation" on page 181

Restarting or terminating a load operation in a partitioned database environment

The load process in a multi-partition database consists of two stages: the setup stage where database partition-level resources such as table locks on output database partitions are acquired, and the load stage where data is formatted and loaded into tables on the database partitions. The four database partition error isolation modes (LOAD_ERRS_ONLY, SETUP_ERRS_ONLY, SETUP_AND_LOAD_ERRS, and NO_ISOLATION) affect the behavior of load restart and terminate operations when there are errors during one or both of these stages. In general, if a failure occurs during the setup stage, restart and terminate operations are not necessary. However, a failure during the load stage requires a LOAD RESTART or a LOAD TERMINATE on all database partitions involved in the load operation.

Failures During the Setup Stage

When a load operation fails on at least one database partition during the setup stage and the setup stage errors are not being isolated (that is, the error isolation mode is either LOAD_ERRS_ONLY or NO_ISOLATION), the entire load operation is aborted and the state of the table on each database partition is rolled back to the state it was in prior to the load operation. In this case, there is no need to issue a LOAD RESTART or LOAD TERMINATE command.

When a load operation fails on at least one database partition during the initial setup stage and setup stage errors are being isolated (that is, the error isolation mode is either SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS), the load operation continues on the database partitions where the setup stage was successful, but the table on each of the failing database partitions is rolled back to the state it was in prior to the load operation. In this case, there is no need to perform a load restart or terminate operation, unless there is also a failure during the load stage.

To complete the load process on the database partitions where the load operation failed during the setup stage, issue a LOAD REPLACE or LOAD INSERT command and use the OUTPUT_DBPARTNUMS option to specify only the database partition numbers of the database partitions that failed during the original load operation.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del replace into table1 partitioned db config
    isolate_part_errs setup_and_load_errs
```

During the set up stage of the load operation there is a failure on database partitions 1 and 3. Since setup stage errors are isolated, the load operation completes successfully and data is loaded on database partitions 0 and 2. To complete the load operation by loading data on database partitions 1 and 3, issue the following command:

```
load from load.del of del replace into table1 partitioned db config
    output_dbpartnums (1, 3)
```

Failures during the load stage

If a load operation fails on at least one database partition during the load stage of a multi-partition database load operation, a LOAD RESTART or LOAD TERMINATE command must be issued on all database partitions involved in the load operation whether or not they encountered an error while loading data. This is necessary because loading data in a multi-partition database is done through a single transaction. If a load restart operation is initiated, the load operation continues where it left off on all database partitions.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load1.del of del replace into table1 partitioned db config
    isolate_part_errs no_isolation
```

During the load stage of the load operation there is a failure on database partitions 1 and 3. To resume the load operation, the LOAD RESTART command must specify the same set of output database partitions as the original command since the load operation must be restarted on all database partitions:

```
load from load.del of del restart into table1 partitioned db config
    isolate_part_errs no_isolation
```

Note: For load restart operations, the options specified in the LOAD RESTART command will be honored, so it is important that they are identical to the ones specified in the original LOAD command.

If a LOAD TERMINATE command is used when a multi-partition database load operation fails during the load stage, all work done in the previous load operation is lost and the table on each database partition is returned to the state it was in prior to the initial load operation.

For example, table TABLE1 is defined on database partitions 0 through 3 in database WSDB. The following command is issued:

```
load from load.del of del replace into table1 partitioned db config
    isolate_part_errs no_isolation
```

If a failure occurs during the load stage, the load operation can be terminated by issuing a LOAD TERMINATE command that specifies the same output parameters as the original command:

```
load from load.del of del terminate into table1 partitioned db config
    isolate_part_errs no_isolation
```

Related concepts:

- "Load in a partitioned database environment - overview" on page 217
- "Restarting an interrupted load operation" on page 129

Load configuration options for partitioned database environments

PART_FILE_LOCATION X

In the PARTITION_ONLY, LOAD_ONLY, and LOAD_ONLY_VERIFY_PART modes, this parameter can be used to specify the location of the distributed files. This location must exist on each database partition specified by the OUTPUT_DBPARTNUMS option. If the location specified is a relative path name, the path is appended to the current directory to create the location for the distributed files.

For the CURSOR file type, this option must be specified, and the location must refer to a fully qualified file name. This name is the fully qualified base file name of the distributed files that are created on each output database partition in the PARTITION_ONLY mode, or the location of the files to be read from for each database partition in the LOAD_ONLY mode. When using the PARTITION_ONLY mode, multiple files can be created with the specified base name if the target table contains LOB columns.

For file types other than CURSOR, if this option is not specified, the current directory is used for the distributed files.

OUTPUT_DBPARTNUMS X

`X` represents a list of database partition numbers. The database partition numbers represent the database partitions on which the load operation is to be performed. The database partition numbers must be a subset of the database partitions on which the table is defined. All database partitions are selected by default. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)).

PARTITIONING_DBPARTNUMS X

`X` represents a list of database partition numbers that are used in the distribution process. The list must be enclosed in parentheses and the items in the list must be separated by commas. Ranges are permitted (for example, (0, 2 to 10, 15)). The database partitions specified for the distribution process can be different from the database partitions being loaded. If not specified, the LOAD command determines how many database partitions are needed and which database partitions to use in order to achieve optimal performance.

If the ANYORDER modifier is not specified in the LOAD command, only one partitioning agent is used in the load session. Further, if there is only one database partition specified for the OUTPUT_DBPARTNUMS option, or the coordinator partition of the load operation is not an element of OUTPUT:DBPARTNUMS, the coordinator partition of the load operation is used in the distribution process. Otherwise, the first database partition (not the coordinator partition) in OUTPUT_DBPARTNUMS is used in the distribution process.

If the ANYORDER modifier is specified, the number of database partitions used in the distribution process is determined as follows: (number of partitions in OUTPUT_DBPARTNUMS)/4 + 1. Then, the number of database partitions used in the distribution process is chosen from the OUTPUT_DBPARTNUMS, excluding the database partition being used to load the data.

MODE X

Specifies the mode in which the load operation occurs when loading a multi-partition database. PARTITION_AND_LOAD is the default. Valid values are:

- 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. For file types other than CURSOR, the format of the output file name on each database partition is `filename.xxx`, where `filename` is the input file name specified in the LOAD command and `xxx` is the 3-digit database partition number. For the CURSOR file type, the name of the output file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Notes:

  1. This mode cannot be used for a CLI load operation.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the *identityoverride* modifier is specified.
  3. Distribution files generated for file type CURSOR are not compatible between DB2 releases. This means that distribution files of file type CURSOR that were generated in a previous release cannot be loaded using the LOAD_ONLY mode. Similarly, distribution files of file type CURSOR that were generated in the current release cannot be loaded in a future release using the LOAD_ONLY mode.
- 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 CURSOR, the format of the input file name for each database partition should be `filename.xxx`, where `filename` is the name of the file specified in the LOAD command and `xxx` is the 3-digit database partition number. For the CURSOR file type, the name of the input file on each database partition is determined by the PART_FILE_LOCATION option. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

  Notes:

  1. This mode cannot be used for a CLI load operation, or when the CLIENT option of LOAD command is specified.
  2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the *identityoverride* modifier is specified.
- LOAD_ONLY_VERIFY_PART. Data is assumed to be already distributed, but the data file does not contain a partition header. The distributing 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 is written to the load message file for that database partition. The format of the input file name for each

database partition should be `filename.xxx`, where `filename` is the name of the file specified in the LOAD command and `xxx` is the 3-digit database partition number. See the PART_FILE_LOCATION option for details on how to specify the location of the distribution file for each database partition.

Notes:

1. This mode cannot be used for a CLI load operation, or when the CLIENT option of LOAD command is specified.
2. If the table contains an identity column that is needed for distribution, then this mode is not supported, unless the *identityoverride* modifier is specified.

- ANALYZE. An optimal distribution map with even distribution across all database partitions is generated.

MAX_NUM_PART_AGENTS X
Specifies the maximum numbers of partitioning agents to be used in a load session. The default 25.

ISOLATE_PART_ERRS X
Indicates how the load operation reacts to errors that occur on individual database partitions. The default is LOAD_ERRS_ONLY, unless both the ALLOW READ ACCESS and COPY YES options of the LOAD command are specified, in which case the default is NO_ISOLATION. Valid values are:

- SETUP_ERRS_ONLY. 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, 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 cause the entire operation to fail and roll back to the last point of consistency on each database partition.
- LOAD_ERRS_ONLY. Errors that occur on a database partition during setup cause the entire load operation to fail. When an error occurs while data is being loaded the database partitions with errors is rolled back to their last point of consistency. The load operation continues 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 is not visible following the load operation. Because of the errors in the other database partitions the transaction are aborted. Data on all of the database partitions remains invisible until a load restart operation is performed. This makes the newly loaded data visible on the database partitions where the load operation completes and resumes the load operation on database partitions that experienced an error.

  Note: This mode cannot be used when both the ALLOW READ ACCESS and the COPY YES options of the LOAD command are specified.
- 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 LOAD_ERRS_ONLY mode, when partition errors do occur while data is loaded, the data on all database partitions remains invisible until a load restart operation is performed.

Note: This mode cannot be used when both the ALLOW READ ACCESS and the COPY YES options of the LOAD command are specified.

- NO_ISOLATION. Any error during the load operation causes the transaction to fail.

STATUS_INTERVAL X
`X` represents how often you are notified of the volume of data that has been read. The unit of measurement is megabytes (MB). The default is 100 MB. Valid values are whole numbers from 1 to 4000.

PORT_RANGE X
`X` represents the range of TCP ports used to create sockets for internal communications. The default range is from 6000 to 6063. If defined at the time of invocation, the value of the DB2ATLD_PORTS DB2 registry variable replaces the value of the PORT_RANGE load configuration option. For the DB2ATLD_PORTS registry variable, the range should be provided in the following format:

```
<lower-port-number>:<higher-port-number>
```

From the CLP, the format is:

```
( lower-port-number, higher-port-number )
```

CHECK_TRUNCATION
Specifies that the program should check for truncation of data records at input/output. The default behavior is that data is not checked for truncation at input/output.

MAP_FILE_INPUT X
`X` specifies the input file name for the distribution map. This parameter must be specified if the distribution map is customized, as it points to the file containing the customized distribution map. A customized distribution map can be created by using the **db2gpmap** program to extract the map from the database system catalog table, or by using the ANALYZE mode of the LOAD command to generate an optimal map. The map generated by using the ANALYZE mode must be moved to each database partition in your database before the load operation can proceed.

MAP_FILE_OUTPUT X
X represents the output filename for the distribution map. The output file is created on the database partition issuing the LOAD command assuming that database partition is participating in the database partition group where partitioning is performed. If the LOAD command is invoked on a database partition that is not participating in partitioning (as defined by PARTITIONING_DBPARTNUMS()), the output file is created at the first database partition defined with the PARTITIONING_DBPARTNUMS parameter. Consider the following partitioned database environment set-up:

```
1 serv1 0
2 serv1 1
3 serv2 0
4 serv2 1
5 serv3 0
```

Running the following LOAD command on serv3, creates the distribution map on serv1.

```
LOAD FROM file OF ASC METHOD L ( ...) INSERT INTO table CONFIG
MODE ANALYZE PARTITIONING_DBPARTNUMS(1,2,3,4)
MAP_FILE_OUTPUT '/home/db2user/distribution.map'
```

This parameter should be used when the ANALYZE mode is specified. An optimal distribution map with even distribution across all database partitions is generated. If this modifier is not specified and the ANALYZE mode is specified, the program exits with an error.

TRACE X
Specifies the number of records to trace when you require a review of a dump of the data conversion process and the output of the hashing values. The default is 0.

NEWLINE
Used when the input data file is an ASC file with each record delimited by a new line character and the RecLen parameter of the LOAD command is specified. When this option is specified, each record is checked for a new line character. The record length, as specified in the RecLen parameter, is also checked.

DISTFILE X
If this option is specified, the LOAD utility generates a database partition distribution file with the given name. The database partition distribution file contains 4096 integers: one for each entry in the target table's distribution map. Each integer in the file represents the number of rows in the input files being loaded that hashed to the corresponding distribution map entry. This information can help you identify skew in your data and also help you decide whether a new distribution map should be generated for the table using the ANALYZE mode of the utility. If this option is not specified, the default behaviour of the Load utility is to not generate the distribution file.

Note: When this option is specified, a maximum of one distribution agent is used for the load operation. If you explicitly request multiple distribution agents, only one is used.

OMIT_HEADER
Specifies that a distribution map header should not be included in the distribution file. If not specified, a header is generated.

RUN_STAT_DBPARTNUM X
If the STATISTICS YES parameter is specified in the LOAD command, statistics are collected only on one database partition. This parameter specifies on which database partition to collect statistics. If the value is -1 or not specified at all, statistics are collected on the first database partition in the output database partition list.

Related concepts:

- "Moving data using a customized application (user exit)" on page 270

Related tasks:

- "Loading data in a partitioned database environment" on page 219

Related reference:

- "REDISTRIBUTE DATABASE PARTITION GROUP command" in *Command Reference*

Examples of loading data in a partitioned database environment

The following examples demonstrate loading data in a multi-partition database. The database has four database partitions numbered 0 through 3. Database WSDB is defined on all of the database partitions, and table TABLE1 resides in the default database partition group which is also defined on all of the database partitions.

Example 1

To load data into TABLE1 from the user data file `load.del` which resides on database partition 0, connect to database partition 0 and then issue the following command:

```
load from load.del of del replace into table1
```

If the load operation is successful, the output will be as follows:

```
Agent Type     Node     SQL Code     Result
___________________________________________________
LOAD           000      +00000000    Success.
___________________________________________________
LOAD           001      +00000000    Success.
___________________________________________________
LOAD           002      +00000000    Success.
___________________________________________________
LOAD           003      +00000000    Success.
___________________________________________________
PARTITION      001      +00000000    Success.
___________________________________________________
PRE_PARTITION  000      +00000000    Success.
___________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
___________________________________________________

Summary of Partitioning Agents:
Rows Read                    = 100000
Rows Rejected                = 0
Rows Partitioned             = 100000

Summary of LOAD Agents:
Number of rows read          = 100000
Number of rows skipped       = 0
Number of rows loaded        = 100000
Number of rows rejected      = 0
Number of rows deleted       = 0
Number of rows committed     = 100000
```

The output indicates that there was one load agent on each database partition and each ran successfully. It also shows that there was one pre-partitioning agent running on the coordinator partition and one partitioning agent running on database partition 1. These processes completed successfully with a normal SQL return code of 0. The statistical summary shows that the pre-partitioning agent read 100,000 rows, the partitioning agent distributed 100,000 rows, and the sum of all rows loaded by the load agents is 100,000.

Example 2

In the following example, data is loaded into TABLE1 in the PARTITION_ONLY mode. The distributed output files is stored on each of the output database partitions in the directory `/db/data`:

```
load from load.del of del replace into table1 partitioned db config mode
   partition_only part_file_location /db/data
```

The output from the load command is as follows:

```
Agent Type     Node     SQL Code     Result
___________________________________________________
LOAD_TO_FILE   000      +00000000    Success.
___________________________________________________
LOAD_TO_FILE   001      +00000000    Success.
___________________________________________________
LOAD_TO_FILE   002      +00000000    Success.
___________________________________________________
LOAD_TO_FILE   003      +00000000    Success.
___________________________________________________
PARTITION      001      +00000000    Success.
___________________________________________________
PRE_PARTITION  000      +00000000    Success.
___________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 100000
Rows Rejected               = 0
Rows Partitioned            = 100000
```

The output indicates that there was a load-to-file agent running on each output database partition, and these agents ran successfully. There was a pre-partitioning agent on the coordinator partition, and a partitioning agent running on database partition 1. The statistical summary indicates that 100,000 rows were successfully read by the pre-partitioning agent and 100,000 rows were successfully distributed by the partitioning agent. Since no rows were loaded into the table, no summary of the number of rows loaded appears.

Example 3

To load the files that were generated during the PARTITION_ONLY load operation above, issue the following command:

```
load from load.del of del replace into table1 partitioned db config mode
    load_only part_file_location /db/data
```

The output from the load command will be as follows::

```
Agent Type     Node     SQL Code     Result
___________________________________________________
LOAD           000      +00000000    Success.
___________________________________________________
LOAD           001      +00000000    Success.
___________________________________________________
LOAD           002      +00000000    Success.
___________________________________________________
LOAD           003      +00000000    Success.
___________________________________________________
RESULTS:       4 of 4 LOADs completed successfully.
___________________________________________________

Summary of LOAD Agents:
Number of rows read         = 100000
Number of rows skipped      = 0
Number of rows loaded       = 100000
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 100000
```

The output indicates that the load agents on each output database partition ran successfully and that the sum of the number of rows loaded by all load agents is 100,000. No summary of rows distributed is indicated since distribution was not performed.

Example 4 - Failed Load Operation

If the following LOAD command is issued:

```
load from load.del of del replace into table1
```

and one of the loading database partitions runs out of space in the table space during the load operation, the following output is returned:

```
SQL0289N  Unable to allocate new pages in table space "DMS4KT".
SQLSTATE=57011

Agent Type     Node     SQL Code     Result
________________________________________________________________
LOAD           000      +00000000    Success.
________________________________________________________________
LOAD           001      -00000289    Error. May require RESTART.
________________________________________________________________
LOAD           002      +00000000    Success.
________________________________________________________________
LOAD           003      +00000000    Success.
________________________________________________________________
PARTITION      001      +00000000    Success.
________________________________________________________________
PRE_PARTITION  000      +00000000    Success.
________________________________________________________________
RESULTS:       3 of 4 LOADs completed successfully.
________________________________________________________________

Summary of Partitioning Agents:
Rows Read                   = 0
Rows Rejected               = 0
Rows Partitioned            = 0

Summary of LOAD Agents:
Number of rows read         = 0
Number of rows skipped      = 0
Number of rows loaded       = 0
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 0
```

The output indicates that the load operation returned error SQL0289. The database partition summary indicates that database partition 1 ran out of space. Since the default error isolation mode is NO_ISOLATION. the load operation is aborted on all database partitions and a load restart or load terminate operation must be invoked. If additional space is added to the containers of the table space on database partition 1, the load operation can be restarted as follows:

```
load from load.del of del restart into table1
```

Related concepts:

- "Load considerations for partitioned tables" on page 126
- "Load overview" on page 102
- "Loading data in a partitioned database environment - hints and tips" on page 237

Related tasks:

- “Loading data” on page 110
- “Loading data in a partitioned database environment” on page 219

Migration and version compatibility

Loading data in a multi-partition database

You can use the load utility to load data in a multi-partition database.

Reverting to pre-DB2 Universal Database V8 load behavior using the DB2_PARTITIONEDLOAD_DEFAULT registry variable

It is possible to maintain the pre DB2 Universal Database V8 behavior of the LOAD command in a multi-partition database. This allows you to load a file with a valid distribution header into a single database partition without specifying any extra partitioned database configuration options. You can do this by setting the value of the DB2_PARTITIONEDLOAD_DEFAULT registry variable to NO. You may choose to use this option if you want to avoid modifying existing scripts that issue the LOAD command against single database partitions. For example, to load a distribution file into database partition 3 of a table that resides in a database partition group with four database partitions, issue the following command:

```
db2set DB2_PARTITIONEDLOAD_DEFAULT=NO
```

Then issue the following commands from the DB2 Command Line Processor:

```
CONNECT RESET

SET CLIENT CONNECT_NODE 3

CONNECT TO DB MYDB

LOAD FROM LOAD.DEL OF DEL REPLACE INTO TABLE1
```

In a multi-partition database, when no multi-partition database load configuration options are specified, the load operation takes place on all the database partitions on which the table is defined. The input file does not require a distribution header, and the MODE option defaults to PARTITION_AND_LOAD. To load a single database partition, the OUTPUT_DBPARTNUMS option must be specified.

Related reference:
- “LOAD ” on page 132

Loading data in a partitioned database environment - hints and tips

The following is some information to consider before loading a table in a multi-partition database:
- Familiarize yourself with the load configuration options by using the utility with small amounts of data.
- If the input data is already sorted, or in some chosen order, and you want to maintain that order during the loading process, only one database partition should be used for distributing. Parallel distribution cannot guarantee that the data is loaded in the same order it was received. The load utility chooses a single partitioning agent by default if the `anyorder` modifier is not specified on the LOAD command.

- If large objects (LOBs) are being loaded from separate files (that is, if you are using the `lobsinfile` modifier through the load utility), all directories containing the LOB files must be read-accessible to all the database partitions where loading is taking place. The LOAD *lob-path* parameter must be fully qualified when working with LOBs.
- You can force a job running in a multi-partition database to continue even if the load operation detects(at startup time) that some loading database partitions or associated table spaces or tables are offline, by setting the ISOLATE_PART_ERRS option to SETUP_ERRS_ONLY or SETUP_AND_LOAD_ERRS.
- Use the STATUS_INTERVAL load configuration option to monitor the progress of a job running in a multi-partition database. The load operation produces messages at specified intervals indicating how many megabytes of data have been read by the pre-partitioning agent. These messages are dumped to the pre-partitioning agent message file. To view the contents of this file during the load operation, connect to the coordinator partition and issue a LOAD QUERY command against the target table.
- Better performance can be expected if the database partitions participating in the distribution process (as defined by the PARTITIONING_DBPARTNUMS option) are different from the loading database partitions (as defined by the OUTPUT_DBPARTNUMS option), since there is less contention for CPU cycles. When loading data into a multi-partition database, the load utility itself should be invoked on a database partition that is not participating in either the distributing or the loading operation.
- Specifying the MESSAGES parameter in the LOAD command saves the messages files from the pre-partitioning, partitioning, and load agents for reference at the end of the load operation. To view the contents of these files during a load operation, connect to the desired database partition and issue a LOAD QUERY command against the target table.
- The load utility chooses only one output database partition on which to collect statistics. The RUN_STAT_DBPARTNUM database configuration option can be used to specify the database partition.
- Before loading data in a multi-partition database, run the Design Advisor to determine the best partition for each table. For more information, see The Design Advisor.

Troubleshooting

If the load utility is hanging, you can:

- Use the STATUS_INTERVAL parameter to monitor the progress of a multi-partition database load operation. The status interval information is dumped to the pre-partitioning agent message file on the coordinator partition.
- Check the partitioning agent messages file to see the status of the partitioning agent processes on each database partition. If the load is proceeding with no errors, and the TRACE option has been set, there should be trace messages for a number of records in these message files.
- Check the load messages file to see if there are any load error messages.

  Note: You must specify the MESSAGES option of the LOAD command in order for these files to exist.
- Interrupt the current load operation if you find errors suggesting that one of the load processes encountered errors.

Related concepts:

- “Load considerations for partitioned tables” on page 126
- “Load overview” on page 102
- “Monitoring a load operation in a partitioned database environment using the LOAD QUERY command” on page 225

Related tasks:

- “Loading data” on page 110
- “Loading data in a partitioned database environment” on page 219

Related reference:

- “Load configuration options for partitioned database environments” on page 229

Chapter 5. Moving Data Between Systems

This chapter describes how to use the DB2 export, import, and load utilities to transfer data across platforms, and to and from iSeries host databases. The IBM replication tools, used for moving data between databases in an enterprise, are also described.

The following topics are covered:

Moving data across platforms - file format considerations

Compatibility is important when exporting, importing, or loading data across platforms. The following sections describe PC/IXF, delimited ASCII (DEL), and WSF file format considerations when moving data between different operating systems.

PC/IXF File Format

PC/IXF is the recommended file format for transferring data across platforms. PC/IXF files allow the load utility or the import utility to process (normally machine dependent) numeric data in a machine-independent fashion. For example, numeric data is stored and handled differently by Intel® and other hardware architectures.

To provide compatibility of PC/IXF files among all products in the DB2 family, the export utility creates files with numeric data in Intel format, and the import utility expects it in this format.

Depending on the hardware platform, DB2 products convert numeric values between Intel and non-Intel formats (using byte reversal) during both export and import operations.

UNIX based implementations of DB2 database do not create multiple-part PC/IXF files during export. However, they will allow you to import a multiple-part PC/IXF file that was created by DB2. When importing this type of file, all parts should be in the same directory, otherwise an error is returned.

Single-part PC/IXF files created by UNIX based implementations of the DB2 export utility can be imported by DB2 database for Windows.

Delimited ASCII (DEL) File Format

DEL files have differences based on the operating system on which they were created. The differences are:

- Row separator characters
  - UNIX based text files use a line feed (LF) character.
  - Non-UNIX based text files use a carriage return/line feed (CRLF) sequence.
- End-of-file character
  - UNIX based text files do not have an end-of-file character.
  - Non-UNIX based text files have an end-of-file character (X'1A').

Since DEL export files are text files, they can be transferred from one operating system to another. File transfer programs can handle operating system-dependant differences if you transfer the files in text mode; the conversion of row separator and end-of-file characters is not performed in binary mode.

Note: If character data fields contain row separator characters, these will also be converted during file transfer. This conversion causes unexpected changes to the data and, for this reason, it is recommended that you do not use DEL export files to move data across platforms. Use the PC/IXF file format instead.

WSF File Format

Numeric data in WSF format files is stored using Intel machine format. This format allows Lotus® WSF files to be transferred and used in different Lotus operating environments (for example, in Intel based and UNIX based systems).

As a result of this consistency in internal formats, exported WSF files from DB2 products can be used by Lotus 1-2-3® or Symphony running on a different platform. DB2 products can also import WSF files that were created on different platforms.

Transfer WSF files between operating systems in binary (not text) mode.

Note: Do not use the WSF file format to transfer data between DB2 databases on different platforms, because a loss of data can occur. Use the PC/IXF file format instead.

Related reference:

- "Export/Import/Load Utility File Formats" on page 293

Moving XML data

XML data movement overview

With the introduction of an XML column type, support for XML data has been added to the import and export utilities.

Importing XML data:

The import utility can be used to insert XML documents into a regular relational table. Only well-formed XML documents can be imported.

Use the XML FROM option of the IMPORT command to specify the location of the XML documents to import. The XMLVALIDATE option specifies how imported documents should be validated. You can select to have the imported XML data validated against a schema specified with the IMPORT command, against a schema identified by a schema location hint inside of the source XML document, or by the schema identified by the XML Data Specifier in the main data file. You can also use the XMLPARSE option to specify how whitespace should be handled when the XML document is imported. The `xmlchar` and `xmlgraphic` file type modifiers allow you to specify the encoding characteristics for the imported XML data.

Exporting XML data:

Data may be exported from tables that include one or more columns with an XML data type. Exported XML data is stored in files separate from the main data file containing the exported relational data. Information about each exported XML document is represented in the main exported data file by an XML data specifier (XDS). The XDS is a string that specifies the name of the system file in which the XML document is stored, the exact location and length of the XML document inside of this file, and the XML schema used to validate the XML document.

You can use the XMLFILE, XML TO, and XMLSAVESCHEMA parameters of the EXPORT command to specify details about how exported XML documents are stored. The `xmlinsepfiles`, `xmlnodeclaration`, `xmlchar`, and `xmlgraphic` file type modifiers allow you to specify further details about the storage location and the encoding of the exported XML data.

Related concepts:
- "Exporting XML data" on page 5
- "Importing XML data" on page 40
- "Native XML data store overview" in *XML Guide*

Related reference:
- "EXPORT " on page 11
- "IMPORT " on page 49

Important considerations for XML data movement

Following are a number of factors to keep in mind when importing or exporting XML data:
- Exported XML data is always stored separately from the main data file containing exported relational data.
- By default, the export utility writes XML data in Unicode. You can use the `XMLCHAR` file type modifier to have XML data written in the character code page. The `XMLGRAPHIC` file type modifier specifies that XML data is written in the graphic code page, which is UTF-16 regardless of the application code page.
- For the import utility, unless the XML document to import contains a declaration tag that includes an encoding attribute, this document is assumed to be in Unicode. You can use the `XMLCHAR` file type modifier to indicate that XML documents to import are encoded in the character code page, while the `XMLGRAPHIC` file type modifier indicates that XML documents to import are encoded in UTF-16.

- For the import utility, rows which contain documents that are not well-formed will be rejected.
- If the XMLVALIDATE option is specified for the import utility, documents which successfully validate against their matching schema will be annotated with the schema information as they are inserted into a table. Rows containing documents that fail to validate against their matching schema will be rejected.
- You can use the export utility with an XQuery specification to export XQuery Data Model (QDM) instances that are not well-formed XML documents. However, exported XML documents that are not well-formed cannot be imported directly into an XML column, since columns defined with the XML data type can contain only complete XML documents.

Related concepts:
- "Exporting XML data" on page 5
- "Importing XML data" on page 40
- "XML data movement overview" on page 242

XML data specifier

XML data involved in the export and import utilities must be stored in files separate from the main data file. The XML data, however, is represented in the main data file with an XML data specifier (XDS). The XDS is a string represented as an XML tag named ″XDS″, which has attributes that describe information about the actual XML data in the column; such information includes the name of the file that contains the actual XML data, and the offset and length of the XML data within that file. The attributes of the XDS are described below.

FIL The name of the file that contains the XML data.

OFF The byte offset of the XML data in the file named by the FIL attribute, where the offset begins from 0.

LEN The length in bytes of the XML data in the file named by the FIL attribute.

SCH The fully qualified SQL identifier of the XML schema that is used to validate this XML document. The schema and name components of the SQL identifier are stored as the ″OBJECTSCHEMA″ and ″OBJECTNAME″ values, respectively, of the row in the SYSCAT.XSROBJECTS catalog table that corresponds to this XML schema.

The XDS is interpreted as a character field in the data file and is subject to the file format's parsing behavior for character columns. For the delimited ASCII file format (DEL), for example, if the character delimiter is present in the XDS, it must be doubled. The special characters (<, >, &, ′, ") within the attribute values must always be escaped. Consider a FIL attribute with the value: `abc&"def".del`. To include this in a delimited ASCII file, where the character delimiter is the ″ character, this XDS would be included as follows: `<XDS FIL=""abc&amp;&quot;def &quot;.del"" />` where the ″ characters are doubled and special characters are escaped.

Example:

The following is an example of an XDS as it would appear in a delimited ASCII data file.

```
"<XDS FIL = ""xmldocs.xml.001"" OFF=""100"" LEN=""300"" />"
```

This entry indicates that the XML data is stored in the file xmldocs.xml.001 beginning at byte offset 100 with a length of 300 bytes. (Because this XDS is within an ASCII file delimited with double quotation marks, the double quotation marks within the XDS tag itself must be doubled.)

Related concepts:
- "Export Overview" on page 1
- "Exporting XML data" on page 5
- "Import Overview" on page 35
- "Importing XML data" on page 40

Related tasks:
- "Exporting data" on page 4
- "Importing data" on page 38

XQuery data model

XML data can be accessed in a database table either by use of the XQuery functions available in SQL, or by invoking XQuery directly. An instance of the XQuery data model (XDM) can be a well-formed XML document, a sequence of nodes, a sequence of atomic values, or any combination of nodes and atomic values.

Individual QDM instances can be written to one or more XML files by means of the EXPORT command.

Related concepts:
- "XML data movement overview" on page 242
- "XML data type" in *XML Guide*

Moving data with DB2 Connect

If you are working in a complex environment in which you need to move data between a host database system and a workstation, you can use DB2 Connect, the gateway for data transfer between the host and the workstation (see Figure 11 on page 246).

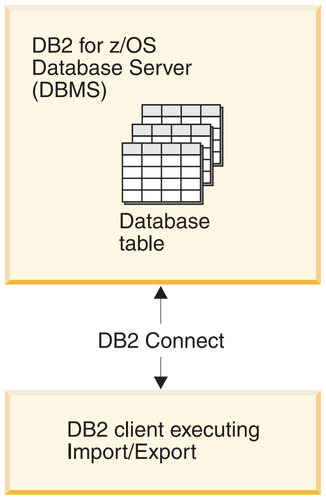


Figure 11. Import/Export through DB2 Connect

The DB2 export and import utilities allow you to move data from a host or iSeries server database to a file on the DB2 Connect workstation, and the reverse. You can then use the data with any other application or relational database management system that supports this export or import format. For example, you can export data from a host or iSeries server database into a PC/IXF file, and then import it into a DB2 for Windows database.

You can perform export and import operations from a database client or from the DB2 Connect workstation.

Notes:

1. The data to be exported or imported must comply with the size and data type restrictions that are applicable to both databases.
2. To improve import performance, you can use compound queries. Specify the `compound` file type modifier in the import utility to group a specified number of query statements into a block. This can reduce network overhead and improve response time.

Restrictions:

With DB2 Connect, export and import operations must meet the following conditions:

- The file type must be PC/IXF.
- A target table with attributes that are compatible with the data must be created on the target server before you can import to it. The **db2look** utility can be used to get the attributes of the source table. Import through DB2 Connect cannot create a table, because INSERT is the only supported option.

If any of these conditions is not met, the operation fails, and an error message is returned.

Note: Index definitions are not stored on export or used on import.

If you export or import mixed data (columns containing both single-byte and double-byte data), consider the following:

- On systems that store data in EBCDIC (MVS™, OS/390®, OS/400®, VM, and VSE), shift-out and shift-in characters mark the start and the end of double-byte data. When you define column lengths for your database tables, be sure to allow enough room for these characters.
- Variable-length character columns are recommended, unless the column data has a consistent pattern.

Moving Data from a workstation to a host server:

To move data to a host or AS/400® and iSeries server database:

1. Export the data from a DB2 table to a PC/IXF file.
2. Using the INSERT option, import the PC/IXF file into a compatible table in the host server database.

To move data from a host server database to a workstation:

1. Export the data from the host server database table to a PC/IXF file.
2. Import the PC/IXF file into a DB2 table.

Example

The following example illustrates how to move data from a workstation to a host or AS/400 and iSeries server database.

1. Export the data into an external IXF format by issuing the following command:
   ```
   db2 export to staff.ixf of ixf select * from userid.staff
   ```
2. Issue the following command to establish a DRDA® connection to the target DB2 database:
   ```
   db2 connect to cbc664 user admin using xxx
   ```
3. If it doesn't already exit, create the target table on the target DB2 database instance_
   ```
   CREATE TABLE mydb.staff (ID SMALLINT NOT NULL, NAME VARCHAR(9),
          DEPT SMALLINT, JOB CHAR(5), YEARS SMALLINT, SALARY DECIMAL(7,2),
          COMM DECIMAL(7,2))
   ```
4. To import the data issue the following command:
   ```
   db2 import from staff.ixf of ixf insert into mydb.staff
   ```

   Each row of data will be read from the file in IXF format, and an SQL INSERT statement will be issued to insert the row into table mydb.staff. Single rows will continue to be inserted until all of the data has been moved to the target table.

Detailed information is available in the following IBM® Redbook: Moving Data Across the DB2 Family. This Redbook can be found at the following URL: http://www.redbooks.ibm.com/redbooks/SG246905.html.

Related concepts:

- "Moving data across platforms - file format considerations" on page 241

Related reference:

- "EXPORT " on page 11
- "IMPORT " on page 49

db2move - Database movement tool

This tool, when used in the EXPORT/IMPORT/LOAD mode, facilitates the movement of large numbers of tables between DB2 databases located on workstations. The tool queries the system catalog tables for a particular database and compiles a list of all user tables. It then exports these tables in PC/IXF format. The PC/IXF files can be imported or loaded to another local DB2 database on the same system, or can be transferred to another workstation platform and imported or loaded to a DB2 database on that platform. Tables with structured type columns are not moved when this tool is used. When used in the COPY mode, this tool facilitates the duplication of a schema.

Authorization:

This tool calls the DB2 export, import, and load APIs, depending on the action requested by the user. Therefore, the requesting user ID must have the correct authorization required by those APIs, or the request will fail.

Command syntax:

```
                                   ┌──────────────────────────┐
                                   ▼                          │
►►──db2move──dbname──action─────────┬─────────────────────────┬┴──────────────────────►◄
                                    ├─-tc──table-definers─────┤
                                    ├─-tn──table-names────────┤
                                    ├─-sn──schema-names───────┤
                                    ├─-ts──tablespace-names───┤
                                    ├─-tf──filename───────────┤
                                    ├─-io──import-option──────┤
                                    ├─-lo──load-option────────┤
                                    ├─-co──copy-option────────┤
                                    ├─-l──lobpaths────────────┤
                                    ├─-u──userid──────────────┤
                                    ├─-p──password────────────┤
                                    └─-aw─────────────────────┘
```

Command parameters:

dbname
: Name of the database.

action Must be one of:

EXPORT
: Exports all tables that meet the filtering criteria in `options`. If no `options` are specified, exports all the tables. Internal staging information is stored in the `db2move.lst` file.

IMPORT
: Imports all tables listed in the internal staging file `db2move.lst`. Use the `-io` option for IMPORT specific actions.

LOAD
: Loads all tables listed in the internal staging file `db2move.lst`. Use the `-lo` option for LOAD specific actions.

COPY Duplicates a schema(s) into a target database. Use the `-sn` option to specify one or more schemas. See the `-co` option for COPY specific options. Use the `-tn` or `-tf` option to filter tables in LOAD_ONLY mode.

See below for a list of files that are generated during each action.

-tc table-definers. The default is all definers.

This is an EXPORT action only. If specified, only those tables created by the definers listed with this option are exported. If not specified, the default is to use all definers. When specifying multiple definers, they must be separated by commas; no blanks are allowed between definer IDs. This option can be used with the "-tn" table-names option to select the tables for export.

An asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

-tn table-names. The default is all user tables.

This is an EXPORT or COPY action only. If specified, only those tables whose names match exactly those in the specified string are exported or copied. If not specified, the default is to use all user tables. When specifying multiple table names, they must be separated by commas; no blanks are allowed between table names. When using the COPY action, the table names should be listed with their schema qualifier in the format "schema"."table". When using the EXPORT action, the table names should be listed unqualified. This option can be used with the "-tc" table-definers option to select the tables for export. **db2move** will only act on those tables whose names match the specified table names and whose definers match the specified table definers.

For export, an asterisk (*) can be used as a wildcard character that can be placed anywhere in the string.

-sn schema-names. The default for EXPORT is all schemas (not for COPY).

If specified, only those tables whose schema names match exactly will be exported or copied. If multiple schema names are specified, they must be separated by commas; no blanks are allowed between schema names. Schema names of less than 8 character are padded to 8 characters in length.

In the case of export:

If the asterisk wildcard character (*) is used in the schema names, it will be changed to a percent sign (%) and the table name (with percent sign) will be used in the LIKE predicate of the WHERE clause. If not specified, the default is to use all schemas. If used with the `-tn` or `-tc` option, **db2move** will only act on those tables whose schemas match the specified schema names and whose definers match the specified definers. A schema name ′fred′ has to be specified ″-sn fr*d*″ instead of ″-sn fr*d″ when using an asterisk.

-ts tablespace-names. The default is all table spaces.

This is an EXPORT action only. If this option is specified, only those tables that reside in the specified table space will be exported. If the asterisk wildcard character (*) is used in the table space name, it will be changed to a percent sign (%) and the table name (with percent sign) will be used in the LIKE predicate in the WHERE clause. If the -ts option is not specified, the default is to use all table spaces. If multiple table space names are specified, they must be separated by commas; no blanks are allowed between table space names. Table space names less than 8 characters are padded to 8 characters in length. For example, a table space name ′mytb′ has to be specified ″-ts my*b*″ instead of ″-sn my*b″ when using the asterisk.

-tf filename

This is an EXPORT or COPY action only. If specified, only the tables listed in the given file will be exported or copied. The tables should be listed one per line, and each table should be fully qualified. Here is an example of the contents of a file:

```
"SCHEMA1"."TABLE NAME1"
"SCHEMA NAME77"."TABLE155"
```

-io import-option. The default is REPLACE_CREATE.

Valid options are: INSERT, INSERT_UPDATE, REPLACE, CREATE, and REPLACE_CREATE.

-lo load-option. The default is INSERT.

Valid options are: INSERT and REPLACE.

-co When the **db2move** action is COPY, the following `-co` follow-on options will be available:

“TARGET_DB <db name> [USER <userid> USING <password>]”
Allows the user to specify the name of the target database and the user/password. (The source database user/password can be specified using the existing `-p` and `-u` options). The `USER/USING` clause is optional. If `USER` specifies a userid, then the password must either be supplied following the `USING` clause, or if it’s not specified, then **db2move** will prompt for the password information. The reason for prompting is for security reasons discussed below. `TARGET_DB` is a mandatory option for the COPY action. The `TARGET_DB` cannot be the same as the source database. The `ADMIN_COPY_SCHEMA` procedure can be used for copying schemas within the same database. The `COPY` action requires inputting at least one schema (`-sn`) or one table (`-tn` or `-tf`).

Running multiple **db2move** commands to copy schemas from one database to another will result in deadlocks. Only one **db2move** command should be issued at a time. Changes to tables in the source schema during copy processing may mean that the data in the target schema is not identical following a copy.

“MODE”

DDL_AND_LOAD
Creates all supported objects from the source schema, and populates the tables with the source table data. This is the default option.

DDL_ONLY
Creates all supported objects from the source schema, but does not repopulate the tables.

LOAD_ONLY
Loads all specified tables from the source database to the target database. The tables must already exist on the target.

This is an optional option that is only used with the COPY action.

"SCHEMA_MAP"
Allows user to rename schema when copying to target. Provides a list of the source-target schema mapping, separated by commas, surrounded by brackets. e.g schema_map ((s1, t1), (s2, t2)). This would mean objects from schema s1 will be copied to schema t1 on the target; objects from schema s2 will be copied to schema t2 on the target. The default, and recommended, target schema name is the source schema name. The reason for this is **db2move** will not attempt to modify the schema for any qualified objects within object bodies. Therefore, using a different target schema name may lead to problems if there are qualified objects within the object body.

For example: `create view FOO.v1 as 'select c1 from FOO.t1'`

In this case, copy of schema FOO to BAR, v1 will be regenerated as: `create view BAR.v1 as 'select c1 from FOO.t1'`

This will either fail since schema FOO does not exist on the target database, or have an unexpected result due to FOO being different than BAR. Maintaining the same schema name as the source will avoid these issues. If there are cross dependencies between schemas, all inter-dependant schemas must be copied or there may be errors copying the objects with the cross dependencies.

For example: `create view FOO.v1 as 'select c1 from BAR.t1'`

In this case, the copy of v1 will either fail if BAR is not copied as well, or have an unexpected result if BAR on the target is different than BAR from the source. **db2move** will not attempt to detect cross schema dependencies.

This is an optional option that is only used with the COPY action.

"NONRECOVERABLE"
This option allows the user to override the default behaviour of the load to be done with COPY-NO. With the default behaviour, the user will be forced to take backups of each tablespace that was loaded into. When specifying this NONRECOVERABLE keyword, the user will not be forced to take backups of the tablespaces immediately. It is, however, highly recommended that the backups be taken as soon as possible to ensure the newly created tables will be properly recoverable. This is an optional option available to the COPY action.

"OWNER"
Allows the user to change the owner of each new object created in the target schema after a successful COPY. The default owner of the target objects will be the connect user; if this option is specified, ownership will be transfered to the new owner. This option is pending due to containability Q1/2006 delivery but this parameter will be in the first design. This is an optional option available to the COPY action.

"TABLESPACE_MAP"
The user may specify tablespace name mappings to be used instead of the tablespaces from the source system during a copy. This will be an array of tablespace mappings surrounded by brackets. For example, `tablespace_map          ((TS1,`

TS2),(TS3, TS4)). This would mean that all objects from tablespace TS1 will be copied into tablespace TS2 on the target database and objects from tablespace TS3 will be copied into tablespace TS4 on the target. In the case of ((T1, T2),(T2, T3)), all objects found in T1 on the source database will be recreated in T2 on the target database and any objects found in T2 on the source database will be recreated in T3 on the target database. The default is to use the same tablespace name as from the source, in which case, the input mapping for this tablespace is not necassary. If the specified tablespace does not exist, the copy of the objects using that tablespace will fail and be logged in the error file.

The user also has the option of using the `SYS_ANY` keyword to indicate that the target tablespace should be chosen using the default tablespace selection algorithm. In this case, **db2move** will be able to chose any available tablespace to be used as the target. The `SYS_ANY` keyword can be used for all tablespaces, example: `tablespace_map SYS_ANY`. In addition, the user can specify specific mappings for some tablespaces, and the default tablespace selection algorithm for the remaining. For example, `tablespace_map ((TS1, TS2),(TS3, TS4), SYS_ANY)`. This indicates that tablespace TS1 is mapped to TS2, TS3 is mapped to TS4, but the remaining tablespaces will be using a default tablespace target. The `SYS_ANY` keyword is being used since it’s not possible to have a tablespace starting with ″SYS″.

This is an optional option available to the COPY action.

-l lobpaths. For IMPORT and EXPORT, if this option is specified, it will be also used for XML paths. The default is the current directory.

This option specifies the absolute path names where LOB or XML files are created (as part of EXPORT) or searched for (as part of IMPORT or LOAD). When specifying multiple paths, each must be separated by commas; no blanks are allowed between paths. If multiple paths are specified, EXPORT will use them in round-robin fashion. It will write one LOB document to the first path, one to the second path, and so on up to the last, then back to the first path. The same is true for XML documents. If files are not found in the first path (during IMPORT or LOAD), the second path will be used, and so on.

-u userid. The default is the logged on user ID.

Both user ID and password are optional. However, if one is specified, the other must be specified. If the command is run on a client connecting to a remote server, user ID and password should be specified.

-p Password. The default is the logged on password. Both user ID and password are optional. However, if one is specified, the other must be specified. When the `-p` option is specified, but the password not supplied, **db2move** will prompt for the password. This is done for security reasons. Inputting the password through command line creates security issues. For example, a `ps -ef` command would display the password. If, however, **db2move** is invoked through a script, then the passwords will have to be supplied. If the command is issued on a client connecting to a remote server, user ID and password should be specified.

-aw Allow Warnings. When ’-aw’ is not specified, tables that experience warnings during export are not included in the `db2move.lst` file (although that table’s .ixf file and .msg file are still generated). In some scenarios (such as data truncation) the user might wish to allow such tables to be included in the `db2move.lst` file. Specifing this option allows tables which receive warnings during export to be included in the .lst file.

Examples:

- To export all tables in the SAMPLE database (using default values for all options), issue:
```
db2move sample export
```
- To export all tables created by `userid1` or user IDs LIKE `us%rid2`, and with the name `tbname1` or table names LIKE `%tbname2`, issue:
```
db2move sample export -tc userid1,us*rid2 -tn tbname1,*tbname2
```
- To import all tables in the SAMPLE database (LOB paths `D:\LOBPATH1` and `C:\LOBPATH2` are to be searched for LOB files; this example is applicable to Windows operating systems only), issue:
```
db2move sample import -l D:\LOBPATH1,C:\LOBPATH2
```
- To load all tables in the SAMPLE database (`/home/userid/lobpath` subdirectory and the `tmp` subdirectory are to be searched for LOB files; this example is applicable to Linux and UNIX-based systems only), issue:
```
db2move sample load -l /home/userid/lobpath,/tmp
```
- To import all tables in the SAMPLE database in REPLACE mode using the specified user ID and password, issue:
```
db2move sample import -io replace -u userid -p password
```
- To duplicate schema `schema1` from source database `dbsrc` to target database `dbtgt`, issue:
```
db2move dbsrc COPY -sn schema1 -co TARGET_DB dbtgt USER myuser1 USING mypass1
```
- To duplicate schema `schema1` from source database `dbsrc` to target database `dbtgt`, rename the schema to `newschema1` on the target, and map source tablespace `ts1` to `ts2` on the target, issue:
```
db2move dbsrc COPY -sn schema1 -co TARGET_DB dbtgt USER myuser1 USING mypass1
     SCHEMA_MAP ((schema1,newschema1)) TABLESPACE_MAP ((ts1,ts2), SYS_ANY))
```

Usage notes:

- Loading data into tables containing XML columns is not supported. The workaround is to manually issue the **IMPORT** or **EXPORT** commands, or use the **db2move -Export** and **db2move -Import** behaviour. If these tables also contain generated always identity columns, data cannot be imported into the tables.
- This tool exports, imports, or loads user-created tables. If a database is to be duplicated from one operating system to another operating system, **db2move** facilitates the movement of the tables. It is also necessary to move all other objects associated with the tables, such as aliases, views, triggers, user-defined functions, and so on. If the import utility with the REPLACE_CREATE option is used to create the tables on the target database, then the limitations outlined in Using import to recreate an exported table are imposed. If unexpected errors are encountered during the **db2move** import phase when the REPLACE_CREATE option is used, examine the appropriate `tabnnn.msg` message file and consider that the errors might be the result of the limitations on table creation.
- When export, import, or load APIs are called by **db2move**, the `FileTypeMod` parameter is set to `lobsinfile`. That is, LOB data is kept in file separate from the PC/IXF file, for every table.
- The LOAD command must be run locally on the machine where the database and the data file reside. When the `load API` is called by **db2move**, the

`NONRECOVERABLE` option is used. If logretain is on, and the `-lo` option is `INSERT`, the load operation marks the table as inaccessible and it must be dropped. The table space where the loaded tables reside is placed in backup pending state, and is not accessible. A full database backup, or a table space backup, is required to take the table space out of backup pending state. Performance for the **DB2MOVE** command with the IMPORT action can be improved by altering the default buffer pool, IBMDEFAULTBP; and by updating the configuration parameters `sortheap`, `util_heap_sz`, `logfilsz`, and `logprimary`.

- For more information on the NONRECOVERABLE recoverability option see the Data Movement Utilities Guide and Reference.

Files Required/Generated When Using EXPORT:

- Input: None.
- Output:

EXPORT.out	The summarized result of the EXPORT action.
db2move.lst	The list of original table names, their corresponding PC/IXF file names (tabnnn.ixf), and message file names (tabnnn.msg). This list, the exported PC/IXF files, and LOB files (tabnnnc.yyy) are used as input to the **db2move** IMPORT or LOAD action.
tabnnn.ixf	The exported PC/IXF file of a specific table.
tabnnn.msg	The export message file of the corresponding table.
tabnnnc.yyy	The exported LOB files of a specific table. “nnn” is the table number. “c” is a letter of the alphabet. “yyy” is a number ranging from 001 to 999. These files are created only if the table being exported contains LOB data. If created, these LOB files are placed in the “lobpath” directories. There are a total of 26 000 possible names for the LOB files.
system.msg	The message file containing system messages for creating or deleting file or directory commands. This is only used if the action is EXPORT, and a LOB path is specified.

Files Required/Generated When Using IMPORT:

- Input:

db2move.lst	An output file from the EXPORT action.
tabnnn.ixf	An output file from the EXPORT action.
tabnnnc.yyy	An output file from the EXPORT action.

- Output:

IMPORT.out	The summarized result of the IMPORT action.
tabnnn.msg	The import message file of the corresponding table.

Files Required/Generated When Using LOAD:

- Input:

db2move.lst	An output file from the EXPORT action.
tabnnn.ixf	An output file from the EXPORT action.
tabnnnc.yyy	An output file from the EXPORT action.

- Output:

LOAD.out	The summarized result of the LOAD action.
tabnnn.msg	The LOAD message file of the corresponding table.

Files Required/Generated When Using COPY:

- Input: None
- Output:

  COPYSCHEMA.msg
  An output file from the COPY action.

  COPYSCHEMA.err
  An output file from the COPY action.

  LOADTABLE.err
  An output file from the COPY action.

  LOADTABLE.msg
  An output file from the COPY action.

  These files are timestamped and all files that are generated from one run will have the same timestamp.

Related reference:

- "db2look - DB2 statistics and DDL extraction tool command" in *Command Reference*

db2relocatedb - Relocate database

This command renames a database, or relocates a database or part of a database (for example, the container and the log directory) as specified in the configuration file provided by the user. This tool makes the necessary changes to the DB2 instance and database support files.

Authorization:

None

Command syntax:

```
►►──db2relocatedb──-f──configFilename────────────────────────────────►◄
```

Command parameters:

-f configFilename
Specifies the name of the file containing the configuration information necessary for relocating the database. This can be a relative or absolute file name. The format of the configuration file is:

```
DB_NAME=oldName,newName
DB_PATH=oldPath,newPath
INSTANCE=oldInst,newInst
NODENUM=nodeNumber
LOG_DIR=oldDirPath,newDirPath
CONT_PATH=oldContPath1,newContPath1
CONT_PATH=oldContPath2,newContPath2
```

```
...
STORAGE_PATH=oldStoragePath1,newStoragePath1
STORAGE_PATH=oldStoragePath2,newStoragePath2
...
```

Where:

DB_NAME
: Specifies the name of the database being relocated. If the database name is being changed, both the old name and the new name must be specified. This is a required field.

DB_PATH
: Specifies the original path of the database being relocated. If the database path is changing, both the old path and new path must be specified. This is a required field.

INSTANCE
: Specifies the instance where the database exists. If the database is being moved to a new instance, both the old instance and new instance must be specified. This is a required field.

NODENUM
: Specifies the node number for the database node being changed. The default is 0.

LOG_DIR
: Specifies a change in the location of the log path. If the log path is being changed, both the old path and new path must be specified. This specification is optional if the log path resides under the database path, in which case the path is updated automatically.

CONT_PATH
: Specifies a change in the location of table space containers. Both the old and new container path must be specified. Multiple CONT_PATH lines can be provided if there are multiple container path changes to be made. This specification is optional if the container paths reside under the database path, in which case the paths are updated automatically. If you are making changes to more than one container where the same old path is being replaced by a common new path, a single `CONT_PATH` entry can be used. In such a case, an asterisk (*) could be used both in the old and new paths as a wildcard.

STORAGE_PATH
: This is only applicable to databases with automatic storage enabled. It specifies a change in the location of one of the storage paths for the database. Both the old storage path and the new storage path must be specified. Multiple STORAGE_PATH lines can be given if there are several storage path changes to be made.

Blank lines or lines beginning with a comment character (#) are ignored.

Usage notes:

If the instance that a database belongs to is changing, the following must be done before running this command to ensure that changes to the instance and database support files are made:

- If a database is being moved to another instance, create the new instance.

- Copy the files and devices belonging to the databases being copied onto the system where the new instance resides. The path names must be changed as necessary. However, if there are already databases in the directory where the database files are moved to, you can mistakenly overwrite the existing `sqldbdir` file, thereby removing the references to the existing databases. In this scenario, the **db2relocatedb** utility cannot be used. Instead of **db2relocatedb**, an alternative is a redirected restore operation.
- Change the permission of the files/devices that were copied so that they are owned by the instance owner.

If the instance is changing, the tool must be run by the new instance owner.

In a partitioned database environment, this tool must be run against every database partition that requires changes. A separate configuration file must be supplied for each database partition, that includes the NODENUM value of the database partition being changed. For example, if the name of a database is being changed, every database partition will be affected and the **db2relocatedb** command must be run with a separate configuration file on each database partition. If containers belonging to a single database partition are being moved, the **db2relocatedb** command only needs to be run once on that database partition.

Examples:

Example 1

To change the name of the database TESTDB to PRODDB in the instance db2inst1 that resides on the path `/home/db2inst1`, create the following configuration file:

```
DB_NAME=TESTDB,PRODDB
DB_PATH=/home/db2inst1
INSTANCE=db2inst1
NODENUM=0
```

Save the configuration file as `relocate.cfg` and use the following command to make the changes to the database files:

```
db2relocatedb -f relocate.cfg
```

Example 2

To move the database DATAB1 from the instance jsmith on the path `/dbpath` to the instance prodinst do the following:

1. Move the files in the directory `/dbpath/jsmith` to `/dbpath/prodinst`.
2. Use the following configuration file with the **db2relocatedb** command to make the changes to the database files:

   ```
   DB_NAME=DATAB1
   DB_PATH=/dbpath
   INSTANCE=jsmith,prodinst
   NODENUM=0
   ```

Example 3

The database PRODDB exists in the instance inst1 on the path `/databases/PRODDB`. The location of two table space containers needs to be changed as follows:

- SMS container `/data/SMS1` needs to be moved to `/DATA/NewSMS1`.
- DMS container `/data/DMS1` needs to be moved to `/DATA/DMS1`.

After the physical directories and files have been moved to the new locations, the following configuration file can be used with the **db2relocatedb** command to make changes to the database files so that they recognize the new locations:

```
DB_NAME=PRODDB
DB_PATH=/databases/PRODDB
INSTANCE=inst1
NODENUM=0
CONT_PATH=/data/SMS1,/DATA/NewSMS1
CONT_PATH=/data/DMS1,/DATA/DMS1
```

Example 4

The database TESTDB exists in the instance db2inst1 and was created on the path /databases/TESTDB. Table spaces were then created with the following containers:

```
TS1
TS2_Cont0
TS2_Cont1
/databases/TESTDB/TS3_Cont0
/databases/TESTDB/TS4/Cont0
/Data/TS5_Cont0
/dev/rTS5_Cont1
```

TESTDB is to be moved to a new system. The instance on the new system will be newinst and the location of the database will be /DB2.

When moving the database, all of the files that exist in the /databases/TESTDB/db2inst1 directory must be moved to the /DB2/newinst directory. This means that the first 5 containers will be relocated as part of this move. (The first 3 are relative to the database directory and the next 2 are relative to the database path.) Since these containers are located within the database directory or database path, they do not need to be listed in the configuration file. If the 2 remaining containers are to be moved to different locations on the new system, they must be listed in the configuration file.

After the physical directories and files have been moved to their new locations, the following configuration file can be used with **db2relocatedb** to make changes to the database files so that they recognize the new locations:

```
DB_NAME=TESTDB
DB_PATH=/databases/TESTDB,/DB2
INSTANCE=db2inst1,newinst
NODENUM=0
CONT_PATH=/Data/TS5_Cont0,/DB2/TESTDB/TS5_Cont0
CONT_PATH=/dev/rTS5_Cont1,/dev/rTESTDB_TS5_Cont1
```

Example 5

The database TESTDB has two database partitions on database partition servers 10 and 20. The instance is servinst and the database path is /home/servinst on both database partition servers. The name of the database is being changed to SERVDB and the database path is being changed to /databases on both database partition servers. In addition, the log directory is being changed on database partition server 20 from /testdb_logdir to /servdb_logdir.

Since changes are being made to both database partitions, a configuration file must be created for each database partition and **db2relocatedb** must be run on each database partition server with the corresponding configuration file.

On database partition server 10, the following configuration file will be used:

```
DB_NAME=TESTDB,SERVDB
DB_PATH=/home/servinst,/databases
INSTANCE=servinst
NODE_NUM=10
```

On database partition server 20, the following configuration file will be used:

```
DB_NAME=TESTDB,SERVDB
DB_PATH=/home/servinst,/databases
INSTANCE=servinst
NODE_NUM=20
LOG_DIR=/testdb_logdir,/servdb_logdir
```

Example 6

The database MAINDB exists in the instance `maininst` on the path `/home/maininst`. The location of four table space containers needs to be changed as follows:

```
/maininst_files/allconts/C0 needs to be moved to /MAINDB/C0
/maininst_files/allconts/C1 needs to be moved to /MAINDB/C1
/maininst_files/allconts/C2 needs to be moved to /MAINDB/C2
/maininst_files/allconts/C3 needs to be moved to /MAINDB/C3
```

After the physical directories and files are moved to the new locations, the following configuration file can be used with the **db2relocatedb** command to make changes to the database files so that they recognize the new locations.

A similar change is being made to all of the containters; that is, `/maininst_files/allconts/` is being replaced by `/MAINDB/` so that a single entry with the wildcard character can be used:

```
DB_NAME=MAINDB
DB_PATH=/home/maininst
INSTANCE=maininst
NODE_NUM=0
CONT_PATH=/maininst_files/allconts/*, /MAINDB/*
```

Related reference:

- "db2inidb - Initialize a mirrored database command" in *Command Reference*

Delimiter restrictions for moving data

Delimiter restrictions:

It is the user's responsibility to ensure that the chosen delimiter character is not part of the data to be moved. If it is, unexpected errors might occur. The following restrictions apply to column, string, DATALINK, and decimal point delimiters when moving data:

- Delimiters are mutually exclusive.
- A delimiter cannot be binary zero, a line-feed character, a carriage-return, or a blank space.
- The default decimal point (.) cannot be a string delimiter.
- The following characters are specified differently by an ASCII-family code page and an EBCDIC-family code page:
  - The Shift-In (0x0F) and the Shift-Out (0x0E) character cannot be delimiters for an EBCDIC MBCS data file.
  - Delimiters for MBCS, EUC, or DBCS code pages cannot be greater than 0x40, except the default decimal point for EBCDIC MBCS data, which is 0x4b.

- Default delimiters for data files in ASCII code pages or EBCDIC MBCS code pages are:
```
" (0x22, double quotation mark; string delimiter)
, (0x2c, comma; column delimiter)
```
- Default delimiters for data files in EBCDIC SBCS code pages are:
```
" (0x7F, double quotation mark; string delimiter)
, (0x6B, comma; column delimiter)
```
- The default decimal point for ASCII data files is 0x2e (period).
- The default decimal point for EBCDIC data files is 0x4B (period).
- If the code page of the server is different from the code page of the client, it is recommended that the hex representation of non-default delimiters be specified. For example,
```
db2 load from ... modified by chardel0x0C coldelX1e ...
```

The following information about support for double character delimiter recognition in DEL files applies to the export, import, and load utilities:

- Character delimiters are permitted within the character-based fields of a DEL file. This applies to fields of type CHAR, VARCHAR, LONG VARCHAR, or CLOB (except when `lobsinfile` is specified). Any pair of character delimiters found between the enclosing character delimiters is imported or loaded into the database. For example,
```
"What a ""nice"" day!"
```
will be imported as:
```
What a "nice" day!
```
In the case of export, the rule applies in reverse. For example,
```
I am 6" tall.
```
will be exported to a DEL file as:
```
"I am 6"" tall."
```
- In a DBCS environment, the pipe (|) character delimiter is not supported.

Related reference:
- "File type modifiers for the export utility" on page 27
- "File type modifiers for the import utility" on page 87
- "File type modifiers for the load utility" on page 188

Moving data between typed tables

The DB2 export and import utilities can be used to move data out of, and into, typed tables. Typed tables can be in a hierarchy. Data movement across hierarchies can include:
- Movement from one hierarchy to an identical hierarchy.
- Movement from one hierarchy to a sub-section of a larger hierarchy.
- Movement from a sub-section of a large hierarchy to a separate hierarchy.

The IMPORT CREATE option allows you to create both the table hierarchy and the type hierarchy.

Identification of types in a hierarchy is database dependent. This means that in different databases, the same type has a different identifier. Therefore, when moving data between these databases, a mapping of the same types must be done to ensure that the data is moved correctly.

Before each typed row is written out during an export operation, an identifier is translated into an index value. This index value can be any number from one to the number of relevant types in the hierarchy. Index values are generated by numbering each type when moving through the hierarchy in a specific order. This order is called the *traverse order*. It is the order of proceeding top-to-bottom, left-to-right through all of the supertables and subtables in the hierarchy. The traverse order is important when moving data between table hierarchies, because it determines where the data is moved in relation to other data.

One method is to proceed from the top of the hierarchy (or the root table), down the hierarchy (subtables) to the bottom subtable, then back up to its supertable, down to the next "right-most" subtable(s), then back up to next higher supertable, down to its subtables, and so on.

The following figure shows a hierarchy with four valid traverse orders:

- Person, Employee, Manager, Architect, Student.
- Person, Student, Employee, Manager, Architect (this traverse order is marked with the dotted line).
- Person, Employee, Architect, Manager, Student.
- Person, Student, Employee, Architect, Manager.

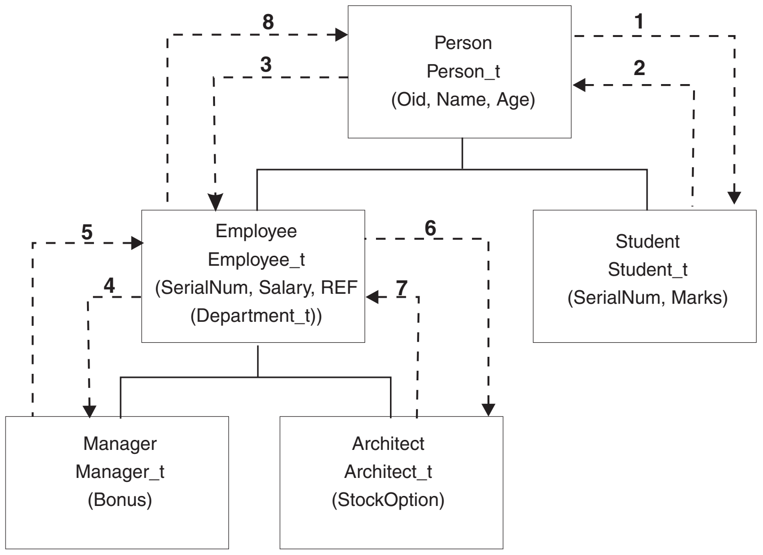


Figure 12.

Related concepts:

- "Export Overview" on page 1
- "Import Overview" on page 35

Moving Data Between Typed Tables - Details

Traverse Order

There is a default traverse order, in which all relevant types refer to all reachable types in the hierarchy from a given starting point in the hierarchy. The default order includes all tables in the hierarchy, and each table is ordered by the scheme used in the OUTER order predicate. There is also a user-specified traverse order, in which the user defines (in a traverse order list) the relevant types to be used. The same traverse order must be used when invoking the export utility and the import utility.

If you are specifying the traverse order, remember that the subtables must be traversed in PRE-ORDER fashion (that is, each branch in the hierarchy must be traversed to the bottom before a new branch is started).

Default Traverse Order

The default traverse order behaves differently when used with different file formats. Assume identical table hierarchy and type relationships in the following:

Exporting data to the PC/IXF file format creates a record of all relevant types, their definitions, and relevant tables. Export also completes the mapping of an index value to each table. During import, this mapping is used to ensure accurate movement of the data to the target database. When working with the PC/IXF file format, you should use the default traverse order.

With the ASC, DEL, or WSF file format, the order in which the typed rows and the typed tables were created could be different, even though the source and target hierarchies might be structurally identical. This results in time differences that the default traverse order will identify when proceeding through the hierarchies. The creation time of each type determines the order taken through the hierarchy at both the source and the target when using the default traverse order. Ensure that the creation order of each type in both the source and the target hierarchies is identical, and that there is structural identity between the source and the target. If these conditions cannot be met, select a user-specified traverse order.

User-Specified Traverse Order

If you want to control the traverse order through the hierarchies, ensure that the same traverse order is used for both the export and the import utilities. Given:

- An identical definition of subtables in both the source and the target databases
- An identical hierarchical relationship among the subtables in both the source and target databases
- An identical traverse order

the import utility guarantees the accurate movement of data to the target database.

Although you determine the starting point and the path down the hierarchy when defining the traverse order, each branch must be traversed to the end before the next branch in the hierarchy can be started. The export and import utilities look for violations of this condition within the specified traverse order.

Related reference:

- "Delimited ASCII (DEL) File Format" on page 294
- "Non-delimited ASCII (ASC) file format" on page 299
- "PC Version of IXF File Format" on page 302
- "Worksheet File Format (WSF)" on page 339

Selection During Data Movement

The movement of data from one hierarchical structure of typed tables to another is done through a specific traverse order and the creation of an intermediate flat file. The export utility (in conjunction with the traverse order) controls what is placed in that file. You only need to specify the target table name and the WHERE clause. The export utility uses these selection criteria to create an appropriate intermediate file.

The import utility controls what is placed in the target database. You can specify an attributes list at the end of each subtable name to restrict the attributes that are moved to the target database. If no attributes list is used, all of the columns in each subtable are moved.

The import utility controls the size and the placement of the hierarchy being moved through the CREATE, INTO table-name, UNDER, and AS ROOT TABLE parameters.

Related reference:
- "IMPORT " on page 49

Examples of Moving Data Between Typed Tables

Examples in this section are based on the following hierarchical structure:

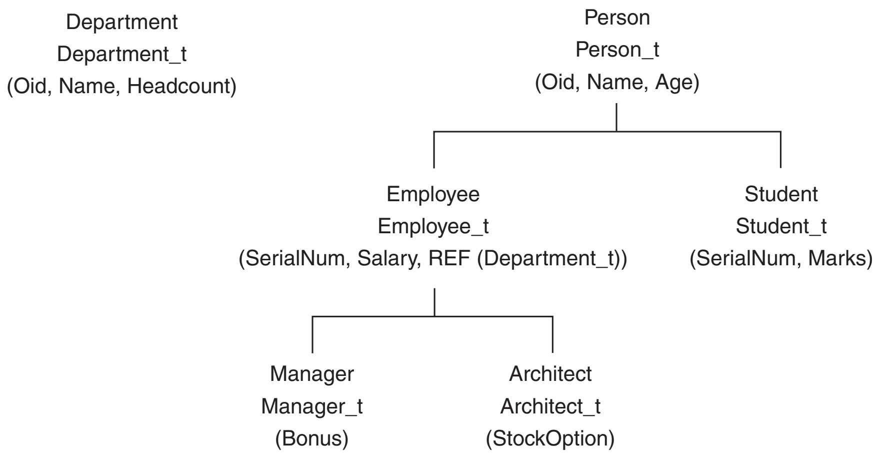


Figure 13.

Example 1

To export an entire hierarchy and then recreate it through an import operation:

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.ixf OF IXF HIERARCHY STARTING Person
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.ixf OF IXF CREATE INTO
   HIERARCHY STARTING Person AS ROOT TABLE
```

Each type in the hierarchy is created if it does not exist. If these types already exist, they must have the same definition in the target database as in the source database. An SQL error (SQL20013N) is returned if they are not the same. Since new hierarchy is being created, none of the subtables defined in the data file being moved to the target database (`Target_db`) can exist. Each of the tables in the source database hierarchy is created. Data from the source database is imported into the correct subtables of the target database.

Example 2

A more complex example shows how to export the entire hierarchy of the source database and import it to the target database. Although all of the data for those people over the age of 20 will be exported, only selected data will be imported to the target database:

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO entire_hierarchy.del OF DEL HIERARCHY (Person,
   Employee, Manager, Architect, Student) WHERE Age>=20
DB2 CONNECT TO Target_db
DB2 IMPORT FROM entire_hierarchy.del OF DEL INSERT INTO (Person,
   Employee(Salary), Architect) IN HIERARCHY (Person, Employee,
   Manager, Architect, Student)
```

The target tables `Person`, `Employee`, and `Architect` must all exist. Data is imported into the `Person`, `Employee`, and `Architect` subtables. That is, the following will be imported:

- All columns in `Person` into `Person`.
- All columns in `Person` plus `Salary` in `Employee` into `Employee`.
- All columns in `Person` plus `Salary` in `Employee`, plus all columns in `Architect` into `Architect`.

Columns `SerialNum` and REF(`Employee_t`) will not be imported into `Employee` or its subtables (that is, `Architect`, which is the only subtable having data imported into it).

Note: Because `Architect` is a subtable of `Employee`, and the only import column specified for `Employee` is `Salary`, `Salary` will also be the only `Employee`-specific column imported into `Architect`. That is, neither `SerialNum` nor REF(`Employee_t`) columns are imported into either `Employee` or `Architect` rows.

Data for the `Manager` and the `Student` tables is not imported.

Example 3

This example shows how to export from a regular table, and import as a single subtable in a hierarchy. The EXPORT command operates on regular (non-typed) tables, so there is no `Type_id` column in the data file. The modifier `no_type_id` is used to indicate this, so that the import utility does not expect the first column to be the `Type_id` column.

```
DB2 CONNECT TO Source_db
DB2 EXPORT TO Student_sub_table.del OF DEL SELECT * FROM
   Regular_Student
DB2 CONNECT TO Target_db
DB2 IMPORT FROM Student_sub_table.del OF DEL METHOD P(1,2,3,5,4)
   MODIFIED BY NO_TYPE_ID INSERT INTO HIERARCHY (Student)
```

In this example, the target table `Student` must exist. Since `Student` is a subtable, the modifier `no_type_id` is used to indicate that there is no `Type_id` in the first column. However, you must ensure that there is an existing `Object_id` column, in addition to all of the other attributes that exist in the `Student` table. `Object-id` is expected to be the first column in each row imported into the `Student` table. The METHOD clause reverses the order of the last two attributes.

Related concepts:

- "Moving data between typed tables" on page 260

Using replication to move data

Replication allows you to copy data on a regular basis to multiple remote databases. If you need to have updates to a master database automatically copied to other databases, you can use the replication features to specify what data should be copied, which database tables the data should be copied to, and how often the updates should be copied. The replication features are part of a larger IBM solution for replicating data in small and large enterprises.

The IBM replication tools are a set of programs and DB2 database tools that copy data between distributed relational database management systems:

- Between DB2 database platforms.
- Between DB2 database platforms and host databases supporting Distributed Relational Database Architecture™ (DRDA) connectivity.
- Between host databases that support DRDA connectivity.

Data can also be replicated to non-IBM relational database management systems by way of Websphere Federation Server.

You can use the IBM replication tools to define, synchronize, automate, and manage copy operations from a single control point for data across your enterprise. The replication tools in IBM DB2 V9.1 offer replication between relational databases. They also work in conjunction with IMS™ DataPropagator™ (formerly DPropNR) to replicate IMS and VSAM data, and with Lotus NotesPump to replicate to and from Lotus Notes® databases.

Replication allows you to give end users and applications access to production data without putting extra load on the production database. You can copy the data to a database that is local to a user or an application, rather than have them access the data remotely. A typical replication scenario involves a source table with copies in one or more remote databases; for example, a central bank and its local branches. At predetermined times, automatic updates of the databases takes place, and all changes to the source database are copied to the target database tables.

The replication tools allow you to customize the copy table structure. You can use SQL when copying to the target database to enhance the data being copied. You can produce read-only copies that duplicate the source table, capture data at a specified point in time, provide a history of changes, or stage data to be copied to additional target tables. Moreover, you can create read-write copies that can be updated by end users or applications, and then have the changes replicated back to the master table, or to peer tables at multiple servers. You can replicate views of source tables, or views of copies. Event-driven replication is also possible.

You can replicate data between DB2 databases on the following platforms: AIX®, iSeries, HP-UX, Linux, Windows, OS/390, SCO UnixWare, Solaris operating system, Sequent®, VM, and VSE. You can also replicate data between DB2 and the following non-DB2 databases: Informix®, Microsoft® Jet, Microsoft SQL Server, Oracle, Sybase, and Sybase SQLAnywhere. In conjunction with other IBM products, you can replicate DB2 data to and from IMS, VSAM, or Lotus Notes. Finally, you can also replicate data to DB2 Everywhere on Windows CE, or Palm OS devices.

Related concepts:

- "The IBM Replication Tools by Component" on page 266

IBM Replication Tools

The IBM Replication Tools by Component

IBM offers two primary replication solutions: Q replication and SQL replication.

The primary components of Q replication are the Q Capture program and the Q Apply program. The primary components of SQL replication are the Capture program and Apply program. Both types of replication share the Replication Alert Monitor tool. You can set up and administer these replication components using the Replication Center and the ASNCLP command-line program.

The following list briefly summarizes these replication components:

Q Capture program:

Reads the DB2 recovery log looking for changes to DB2 source tables and translates committed source data into WebSphere MQ messages that can be published in XML format to a subscribing application, or replicated in a compact format to the Q Apply program.

Q Apply program:

Takes WebSphere MQ messages from a queue, transforms the messages into SQL statements, and updates a target table or stored procedure. Supported targets include DB2 databases or subsystems and Oracle, Sybase, Informix and Microsoft SQL Server databases that are accessed through federated server nicknames.

Capture program:

Reads the DB2 recovery log for changes made to registered source tables or views and then stages committed transactional data in relational tables called change-data (CD) tables, where they are stored until the target system is ready to copy them. SQL replication also provides Capture triggers that populate a staging table called a consistent-change-data (CCD) table with records of changes to non-DB2 source tables.

Apply program:

Reads data from staging tables and makes the appropriate changes to targets. For non-DB2 data sources, the Apply program reads the CCD table through that table's nickname on the federated database and makes the appropriate changes to the target table.

Replication Alert Monitor:

A utility that checks the health of the Q Capture, Q Apply, Capture, and Apply programs. It checks for situations in which a program terminates, issues a warning or error message, reaches a threshold for a specified value, or performs a certain action, and then issues notifications to an email server, pager, or the z/OS console.

Use the Replication Center to:

- Define registrations, subscriptions, publications, queue maps, alert conditions, and other objects.

- Start, stop, suspend, resume, and reinitialize the replication programs.
- Specify the timing of automated copying.
- Specify SQL enhancements to the data.
- Define relationships between the source and the target tables.

Related concepts:
- "Using replication to move data" on page 265

Moving data using the CURSOR file type

By specifying the CURSOR file type when using the LOAD command, you can load the results of an SQL query directly into a target table without creating an intermediate exported file. Additionally, you can load data from another database by referencing a nickname within the SQL query, by using the DATABASE option within the DECLARE CURSOR statement, or by using the sqlu_remotefetch_entry media entry when using the API interface.

There are three approaches for moving data using the CURSOR file type. The first approach uses the Command Line Processor (CLP), the second the API and the third uses the ADMIN_CMD procedure. The key difference between the CLP and the ADMIN_CMD procedure are outlined in the following table.

Table 18. . Differences between the CLP and ADMIN_CMD procedure.

Differences	CLP	ADMIN_CMD_procedure
Syntax	The query statement as well as the source database used by the cursor are defined outside of the LOAD command using a DECLARE CURSOR statement.	The query statement as well as the source database used by the cursor is defined within the LOAD command using the LOAD from (DATABASE database-alias query-statement)
User authorization for accessing a different database	If the data is in a different database than the one you currently connect to, the DATABASE keyword must be used in the DECLARE CURSOR statement. You can specify the user id and password in the same statement as well. If the user id and password are not specified in the DECLARE CURSOR statement, the user id and password explicitly specified for the source database connection are used to access the target database.	If the data is in a different database than the one you are currently connected to, the DATABASE keyword must be used in the LOAD command before the query statement. The user id and password explicitly specified for the source database connection are required to access the target database. You cannot specify a userid or password for the source database. Therefore, if no userid and password were specified when the connection to the target database was made, or the userid and password specified cannot be used to authenticate against the source database, the ADMIN_CMD procedure cannot be used to perform the load.

To execute a LOAD FROM CURSOR operation from the CLP, a cursor must first be declared against an SQL query. Once this is declared, you can issue the LOAD command using the declared cursor's name as the *cursorname* and CURSOR as the file type.

For example:

1. Suppose a source and target table both reside in the same database with the following definitions:

   Table ABC.TABLE1 has 3 columns:
   - ONE INT
   - TWO CHAR(10)
   - THREE DATE

   Table ABC.TABLE2 has 3 columns:
   - ONE VARCHAR
   - TWO INT
   - THREE DATE

   Executing the following CLP commands will load all the data from ABC.TABLE1 into ABC.TABLE2:

   ```
   DECLARE mycurs CURSOR FOR SELECT TWO, ONE, THREE FROM abc.table1
      LOAD FROM mycurs OF cursor INSERT INTO abc.table2
   ```

   Note: The above example shows how to load from an SQL query through the CLP. However, loading from an SQL query can also be accomplished through the **db2Load API**. Define the *piSourceList* of the sqlu_media_list structure to use the *sqlu_statement_entry* structure and SQLU_SQL_STMT media type and define the *piFileType* value as SQL_CURSOR.
2. Suppose the source and target tables reside in different databases with the following definitions:

Table ABC.TABLE1 in database 'dbsource' has 3 columns:
- ONE INT
- TWO CHAR(10)
- THREE DATE

Table ABC.TABLE2 in database 'dbtarget' has 3 columns:
- ONE VARCHAR
- TWO INT
- THREE DATE

You can declare a nickname against the source database, and then declare a cursor against this nickname, and invoke the LOAD command with the FROM CURSOR option, as demonstrated in the following example:

```
<enable federation and define datasource>
 CREATE NICKNAME myschema1.table1 FOR dsdbsource.abc.table1
 DECLARE mycurs CURSOR FOR SELECT TWO,ONE,THREE FROM myschema1.table1
 LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Or, you can use the DATABASE option of the DECLARE CURSOR statement, as demonstrated in the following example:

```
DECLARE mycurs CURSOR DATABASE dbsource FOR SELECT TWO,ONE,THREE FROM abc.table1
LOAD FROM mycurs OF cursor INSERT INTO abc.table2
```

Using the DATABASE option of the DECLARE CURSOR statement (also known as the remotefetch media type when using the Load API) has some benefits over the nickname approach:

Performance

Fetching of data using the remotefetch media type is tightly integrated within a load operation. There are fewer layers of transition to fetch a record compared to the nickname approach. Additionally, when source and target tables are distributed identically in a multi-partition database, the load utility can parallelize the fetching of data, which can further improve performance.

Ease of use

There is no need to enable federation, define a remote datasource, or declare a nickname. Specifying the DATABASE option (and the USER and USING options if necessary) is all that is required.

While this method can be used with cataloged databases, the use of nicknames provides a robust facility for fetching from various data sources which cannot simply be cataloged.

To support this remotefetch functionality, the load utility makes use of infrastructure which supports the SOURCEUSEREXIT facility. The load utility spawns a process which executes as an application to manage the connection to the source database and perform the fetch. This application is associated with its own transaction and is not associated with the transaction under which the load utility is running.

Notes:

1. The previous example shows how to load from an SQL query against a cataloged database through the CLP using the DATABASE option of the DECLARE CURSOR statement. However, loading from an SQL query against a cataloged database can also be done through the **db2Load API**, by defining the *piSourceList* and *piFileTypevalues* of the *db2LoadStruct* structure to use the sqlu_remotefetch_entry media entry and SQLU_REMOTEFETCH media type respectively.
2. As demonstrated in the previous example, the source column types of the SQL query do not need to be identical to their target column types, although they do have to be compatible.

Restrictions:

When loading from a cursor defined using the DATABASE option (or equivalently when using the sqlu_remotefetch_entry media entry with the **db2Load** API), the following restrictions apply:

1. The SOURCEUSEREXIT options cannot be specified concurrently.
2. The METHOD N option is not supported.
3. The USEDEFAULTS option is not supported.

Related tasks:

- "Copying a schema" in *Administration Guide: Implementation*

Related reference:

- "Assignments and comparisons" in *SQL Reference, Volume 1*
- "db2Load - Load data into a table" on page 161
- "ADMIN_CMD procedure – Run administrative commands" in *Administrative SQL Routines and Views*
- "LOAD command using the ADMIN_CMD procedure" on page 145
- "DECLARE CURSOR statement" in *SQL Reference, Volume 2*
- "LOAD " on page 132

Moving data using a customized application (user exit)

The Load SOURCEUSEREXIT option provides a facility through which the load utility can execute a customized script or executable, referred to herein as the user exit. The purpose of the user exit is to populate one or more named pipes with data that is simultaneously read from by the load utility. In a multi-partition databases, multiple instances of the user exit can be invoked concurrently to achieve parallelism of the input data.

As Figure 14 shows, the load utility creates a one or more named pipes and spawns a process to execute your customized executable. Your user exit feeds data into the named pipe(s) while the load utility simultaneously reads.

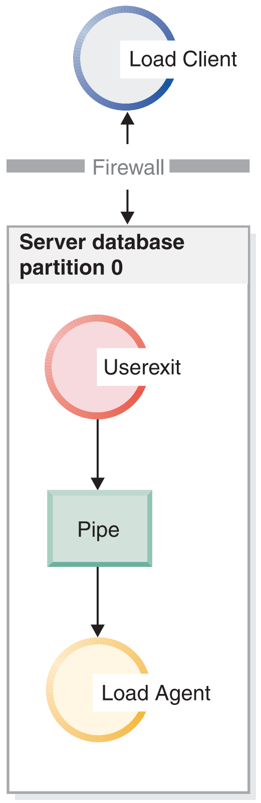


Figure 14. The Load utility reads from the pipe and processes the incoming data.

The data fed into the pipe must reflect the load options specified, including the file type and any file type modifiers. The load utility does not directly read the data files specified. Instead, the data files specified are passed as arguments to your user exit when it is executed.

Invoking your user exit:

The user exit must reside in the bin subdirectory of the DB2 installation directory (often known as sqllib). The load utility invokes the user exit executable with the following command line arguments:

```
<base pipename> <number of source media>
<source media 1> <source media 2> ... <user exit ID>
<number of user exits> <database partition number>
```

Where:

< base pipename >
: Is the base name for named-pipes that the Load utility creates and reads data from. The utility creates one pipe for every source file provided to the LOAD command, and each of these pipes is appended with `.xxx`, where `xxx` is the index of the source file provided. For example, if there are 2 source files provided to the LOAD command, and the <base pipename> argument passed to the user exit is `pipe123`, then the two named pipes that your user exit should feed with data are `pipe123.000` and `pipe123.001`. In a partitioned database environment, the load utility appends the database partition (DBPARTITION) number `.yyy` to the base pipe name, resulting in the pipe name `pipe123.xxx.yyy.`.

<number of source media>
: Is the number of media arguments which follow.

<source media 1> <source media 2> ...
: Is the list of one or more source files specified in the LOAD command. Each source file is placed inside double quotation marks.

<user exit ID>
: Is a special value useful when the PARALLELIZE option is enabled. This integer value (from 1 to N, where N is the total number of user exits being spawned) identifies a particular instance of a running user exit. When the PARALLELIZE option is not enabled, this value defaults to 1.

<number of user exits>
: Is a special value useful when the PARALLELIZE option is enabled. This value represents the total number of concurrently running user exits. When the PARALLELIZE option is not enabled, this value defaults to 1.

<database partition number>
: Is a special value useful when the PARALLELIZE option is enabled. This is the database partition (DBPARTITION) number on which the user exit is executing. When the PARALLELIZE option is not enabled, this value defaults to 0.

Additional options and features:

The following section describes additional SOURCEUSEREXIT facility options:

REDIRECT
: This option allows you to pass data into the STDIN handle or capture data from the STDOUT and STDERR handles of the user exit process.

INPUT FROM BUFFER <buffer>
: Allows you to pass information directly into the STDIN input stream of your user exit. After spawning the process which executes the user exit, the load utility acquires the file-descriptor to the STDIN of this new process and passes in the <buffer> provided. The user exit reads from STDIN to

acquire the information. The load utility simply sends the contents of <buffer> to the user exit using STDIN and does not interpret or modify its contents. For example, if your user exit is designed to read two values from STDIN, an 8 byte user-id and an 8 byte password, your user exit executable written in C might contain the following lines:

```
rc = read (stdin, pUserID, 8);
rc = read (stdin, pPasswd, 8);
```

A user could pass this information using the **INPUT FROM BUFFER** option as shown in the following LOAD command:

```
LOAD FROM myfile1 OF DEL INSERT INTO table1
SOURCEUSEREXIT myuserexit1 REDIRECT INPUT FROM BUFFER myuseridmypasswd
```

Note: The load utility limits the size of the <buffer> to the maximum size of a LOB value. However, from within the command line processor (CLP), the size of the <buffer> is restricted to the maximum size of a CLP statement. From within CLP, it is also recommended that the <buffer> contain only traditional ASCII characters. These issues can be avoided if the load utility is invoked using the db2Load API, or if the INPUT FROM FILE option is used instead.

INPUT FROM FILE <filename>
Allows you to pass the contents of a client side file directly into the STDIN input stream of your user exit. This option is almost identical to the **INPUT FROM BUFFER** option, however this option avoids the potential CLP limitation. The filename must be a fully qualified client side file and must not be larger than the maximum size of a LOB value.

OUTPUT TO FILE < filename>
Allows you to capture the STDOUT and STDERR streams from your user exit process into a server side file. After spawning the process which executes the user exit executable, the load utility redirects the STDOUT and STDERR handles from this new process into the filename specified. This option is useful for debugging and logging errors and activity within your user exit. The filename must be a fully qualified server side file. The filename must be a fully qualified server side file. When the **PARALLELIZE** option is enabled, one file exists per user exit and each file appends a 3 digit numeric identifier, such as *filename.000*.

PARALLELIZE
This option can increase the throughput of data coming into the load utility by invoking multiple user exit processes simultaneously. This option is only applicable to a multi-partition database. The number of user exit instances invoked is equal to the number of distribution agents if data is to be distributed across multiple database partitions during the load operation, otherwise it is equal to the number of loading agents.

The <userexit ID> and <number of userexits> and <database partition number>arguments passed into each user exit reflect the unique identifier (1 to N), the total number of user exits (N), and the database partition (DBPARTITION) number on which the user exit instance is running, respectively. You should ensure that any data written to the named pipe by each user exit process is not duplicated by the other concurrent processes. While there are many ways your user exit application might accomplish this, these values could be helpful to ensure data is not duplicated. For example, if each record of data contains a unique integer column value, your user exit application could use the <userexit ID> and <number

of userexits> values to ensure that each user exit instance returns a unique result set into its named pipe. Your user exit application might use the **MODULUS** property in the following way:

```
i = <userexit ID>
N = <number of user exits>

foreach record
{
   if ((unique-integer MOD N) == i)
   {
     write this record to my named-pipe
   }
}
```

The number of user exit processes spawned depends on the distribution mode specified for database partitioning:

1. As Figure 15 shows, one user exit process is spawned for every distribution-agent when PARTITION_AND_LOAD (default) or PARTITION_ONLY is specified.

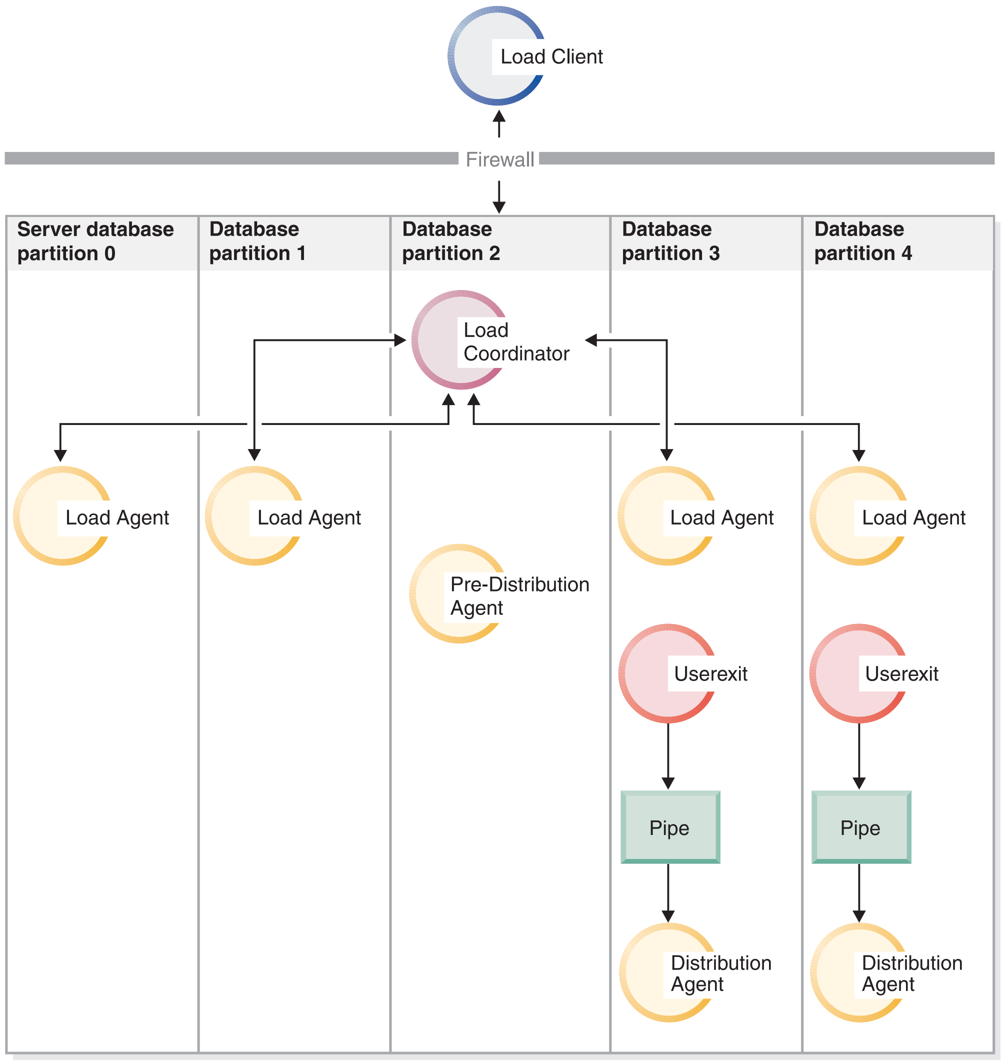


Figure 15. Demonstrates the distrubution mode when PARTITION_AND_LOAD (default) or PARTITION_ONLY is specified.

2. As Figure 16 shows, one user exit process is spawned for every load-agent when LOAD_ONLY or LOAD_ONLY_VERIFY_PART is specified.

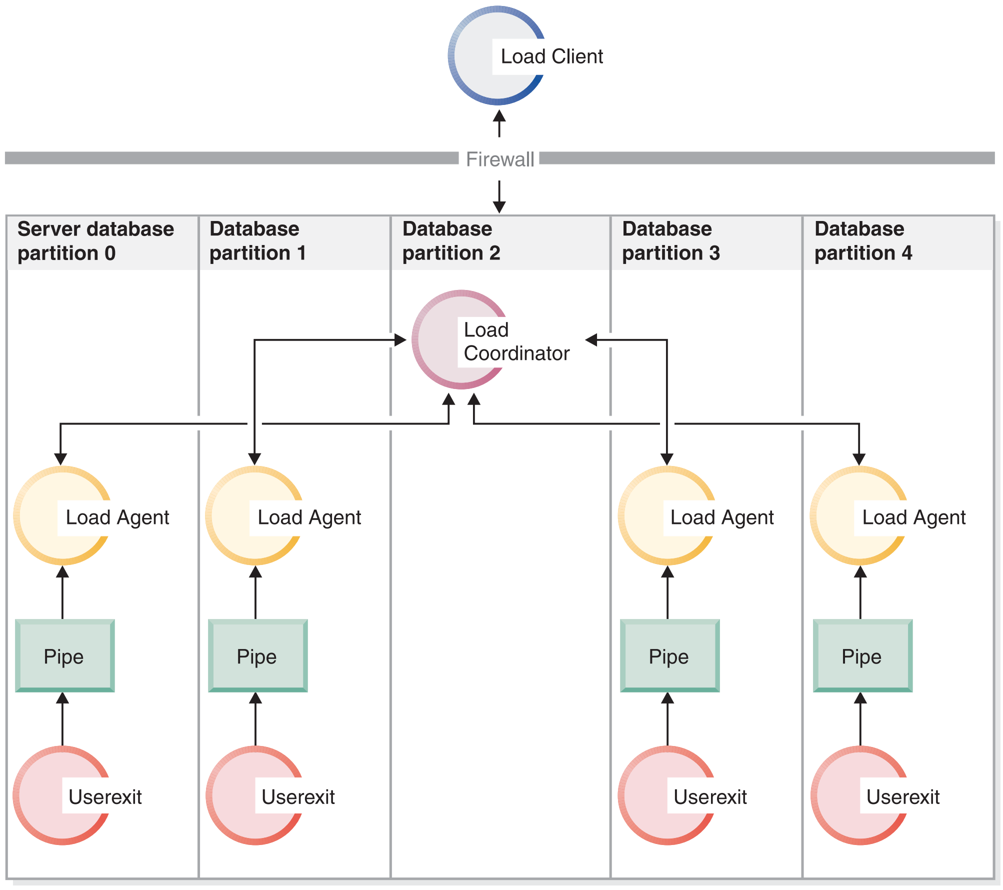


Figure 16. Demonstrates the distrubution mode when LOAD_ONLY or LOAD_ONLY_VERIFY_PART is specified.

Restrictions:

- The LOAD_ONLY and LOAD_ONLY_VERIFY_PART partitioned-db-cfg mode options are not supported when the SOURCEUSEREXIT PARALLELIZE option is not specified.

Related concepts:

- "Load overview" on page 102
- "Loading data in a partitioned database environment - hints and tips" on page 237
- "Moving data using the CURSOR file type" on page 267
- "Schemas" in *SQL Reference, Volume 1*

Related tasks:

- "Copying a schema" in *Administration Guide: Implementation*
- "Restarting a failed copy schema operation" in *Administration Guide: Implementation*

Related reference:

- "LOAD " on page 132

- “Load configuration options for partitioned database environments” on page 229
- “db2Load - Load data into a table” on page 161
- “sqlu_media_list data structure” in *Administrative API Reference*

Appendix A. How to read the syntax diagrams

Throughout this book, syntax is described using the structure defined as follows:

Read the syntax diagrams from left to right and top to bottom, following the path of the line.

The ►►── symbol indicates the beginning of a syntax diagram.

The ──► symbol indicates that the syntax is continued on the next line.

The ►── symbol indicates that the syntax is continued from the previous line.

The ──►◄ symbol indicates the end of a syntax diagram.

Syntax fragments start with the ├── symbol and end with the ──┤ symbol.

Required items appear on the horizontal line (the main path).

```
►►──required_item──────────────────────────────────────────────►◄
```

Optional items appear below the main path.

```
►►──required_item──┬───────────────┬───────────────────────────►◄
                   └─optional_item─┘
```

If an optional item appears above the main path, that item has no effect on execution, and is used only for readability.

```
                   ┌─optional_item─┐
►►──required_item──┴───────────────┴───────────────────────────►◄
```

If you can choose from two or more items, they appear in a stack.

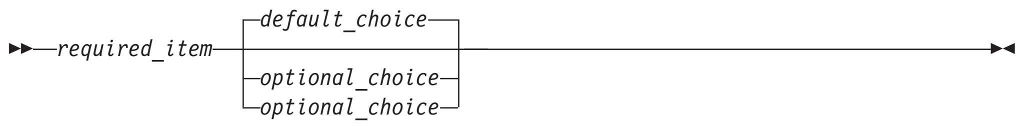
If you *must* choose one of the items, one item of the stack appears on the main path.

```
►►──required_item──┬─required_choice1─┬────────────────────────►◄
                   └─required_choice2─┘
```

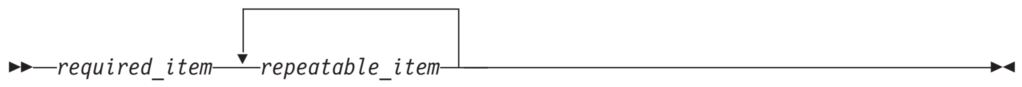
If choosing one of the items is optional, the entire stack appears below the main path.

```
►►──required_item──┬──────────────────┬────────────────────────►◄
                   ├─optional_choice1─┤
                   └─optional_choice2─┘
```

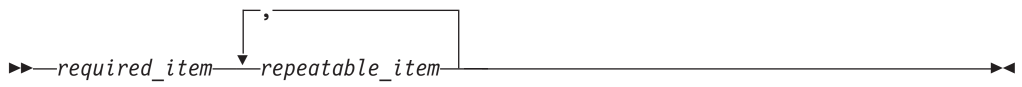
If one of the items is the default, it will appear above the main path, and the remaining choices will be shown below.



An arrow returning to the left, above the main line, indicates an item that can be repeated. In this case, repeated items must be separated by one or more blanks.



If the repeat arrow contains a comma, you must separate repeated items with a comma.

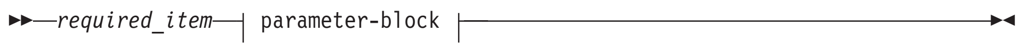


A repeat arrow above a stack indicates that you can make more than one choice from the stacked items or repeat a single choice.

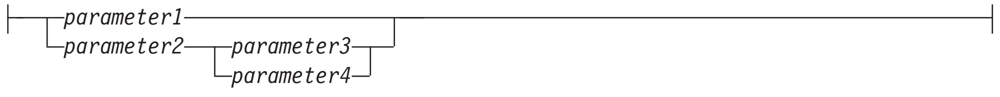
Keywords appear in uppercase (for example, `FROM`). They must be spelled exactly as shown. Variables appear in lowercase (for example, `column-name`). They represent user-supplied names or values in the syntax.

If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

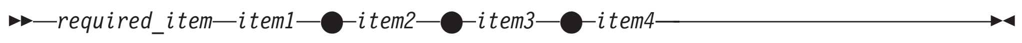
Sometimes a single variable represents a larger fragment of the syntax. For example, in the following diagram, the variable `parameter-block` represents the whole syntax fragment that is labeled **parameter-block**:



parameter-block:



Adjacent segments occurring between "large bullets" (●) may be specified in any sequence.



The above diagram shows that item2 and item3 may be specified in either order. Both of the following are valid:

```
required_item item1 item2 item3 item4
required_item item1 item3 item2 item4
```

Appendix B. Differences between the import and load utility

The following table summarizes the important differences between the DB2 load and import utilities.

Import Utility	**Load Utility**
Slow when moving large amounts of data.	Faster than the import utility when moving large amounts of data, because the load utility writes formatted pages directly into the database.
Limited exploitation of intra-partition parallelism.	Exploitation of intra-partition parallelism. Typically, this requires symmetric multiprocessor (SMP) machines.
No FASTPARSE support.	FASTPARSE support, providing reduced data checking of user-supplied data.
Supports hierarchical data.	Does not support hierarchical data.
Creation of tables, hierarchies, and indexes supported with PC/IXF format.	Tables and indexes must exist.
No support for importing into materialized query tables.	Support for loading into materialized query tables.
WSF format is supported.	WSF format is not supported.
No BINARYNUMERICS support.	BINARYNUMERICS support.
No PACKEDDECIMAL support.	PACKEDDECIMAL support.
No ZONEDDECIMAL support.	ZONEDDECIMAL support.
Cannot override columns defined as GENERATED ALWAYS.	Can override GENERATED ALWAYS columns, by using the GENERATEDOVERRIDE and IDENTITYOVERRIDE file type modifiers.
Supports import into tables, views and nicknames.	Supports loading into tables only.
All rows are logged.	Minimal logging is performed.
Trigger support.	No trigger support.
If an import operation is interrupted, and a *commitcount* was specified, the table is usable and will contain the rows that were loaded up to the last COMMIT. The user can restart the import operation, or accept the table as is.	If a load operation is interrupted, and a *savecount* was specified, the table remains in load pending state and cannot be used until the load operation is restarted, a load terminate operation is invoked, or until the table space is restored from a backup image created some time before the attempted load operation.
Space required is approximately equivalent to the size of the largest index plus 10%. This space is obtained from the temporary table spaces within the database.	Space required is approximately equivalent to the sum of the size of all indexes defined on the table, and can be as much as twice this size. This space is obtained from temporary space within the database.
All constraints are validated during an import operation.	The load utility checks for uniqueness and computes generated column values, but all other constraints must be checked using SET INTEGRITY.

Import Utility	Load Utility
The key values are inserted into the index one at a time during an import operation.	The key values are sorted and the index is built after the data has been loaded.
If updated statistics are required, the runstats utility must be run after an import operation.	Statistics can be gathered during the load operation if all the data in the table is being replaced.
You can import into a host database through DB2 Connect.	You cannot load into a host database.
Import files must exist on the client from which the import utility is invoked.	Depending on the options specified, load files or pipes can reside either on the database partition(s) that contain the database, or on the remotely connected client from which the load utility is invoked.
A backup image is not required. Because the import utility uses SQL inserts, the activity is logged, and no backups are required to recover these operations in case of failure.	A backup image can be created during the load operation.

Related concepts:

- "Import Overview" on page 35
- "Load overview" on page 102

Related reference:

- "IMPORT " on page 49
- "LOAD " on page 132

Appendix C. Export/Import/Load Sessions - API Sample Program

The following sample program shows how to:
- Export data to a file
- Import data to a table
- Load data into a table
- Check the status of a load operation

The source file for this sample program (`tbmove.sqc`) can be found in the `\sqllib\samples\c` directory. It contains both DB2 APIs and embedded SQL calls. The script file `bldapp.cmd`, located in the same directory, contains the commands to build this and other sample programs.

To run the sample program (executable file), enter `tbmove`. You might find it useful to examine some of the generated files, such as the message file, and the delimited ASCII data file.

```
/****************************************************************************
** Licensed Materials - Property of IBM
**
** Governed under the terms of the International
** License Agreement for Non-Warranted Sample Code.
**
** (C) COPYRIGHT International Business Machines Corp. 1996 - 2002
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*****************************************************************************
**
** SOURCE FILE NAME: tbmove.sqc
**
** SAMPLE: How to move table data
**
** DB2 APIs USED:
**         db2Export -- Export
**         db2Import -- Import
**         sqluvqdp -- Quiesce Table Spaces for Table
**         db2Load -- Load
**         db2LoadQuery -- Load Query
**
** SQL STATEMENTS USED:
**         PREPARE
**         DECLARE CURSOR
**         OPEN
**         FETCH
**         CLOSE
**         CREATE TABLE
**         DROP
**
** OUTPUT FILE: tbmove.out (available in the online documentation)
*****************************************************************************
**
** For more information on the sample programs, see the README file.
**
** For information on developing C applications, see the Application
** Development Guide.
```

```
**
** For information on using SQL statements, see the SQL Reference.
**
** For information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, building, and running DB2
** applications, visit the DB2 application development website:
**     http://www.software.ibm.com/data/db2/udb/ad
****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#include <db2ApiDf.h>
#include "utilemb.h"

int DataExport(char *);
int TbImport(char *);
int TbLoad(char *);
int TbLoadQuery(void);

/* support function */
int ExportedDataDisplay(char *);
int NewTableDisplay(void);

EXEC SQL BEGIN DECLARE SECTION;
  char strStmt[256];
  short deptnumb;
  char deptname[15];
EXEC SQL END DECLARE SECTION;

int main(int argc, char *argv[])
{
  int rc = 0;
  char dbAlias[SQL_ALIAS_SZ + 1];
  char user[USERID_SZ + 1];
  char pswd[PSWD_SZ + 1];
  char dataFileName[256];

  /* check the command line arguments */
  rc = CmdLineArgsCheck1(argc, argv, dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  printf("\nTHIS SAMPLE SHOWS HOW TO MOVE TABLE DATA.\n");

  /* connect to database */
  rc = DbConn(dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

#if(defined(DB2NT))
  sprintf(dataFileName, "%s%stbmove.DEL", getenv("DB2PATH"), PATH_SEP);
#else /* UNIX */
  sprintf(dataFileName, "%s%stbmove.DEL", getenv("HOME"), PATH_SEP);
#endif

  rc = DataExport(dataFileName);
  rc = TbImport(dataFileName);
  rc = TbLoad(dataFileName);
  rc = TbLoadQuery();
  /* disconnect from the database */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* main */

int ExportedDataDisplay(char *dataFileName)
{
  struct sqlca sqlca;
  FILE *fp;
  char buffer[100];
  int maxChars = 100;
  int numChars;
  int charNb;

  fp = fopen(dataFileName, "r");
  if (fp == NULL)
  {
    return 1;
  }

  printf("\n  The content of the file '%s' is:\n", dataFileName);
  printf("    ");
  numChars = fread(buffer, 1, maxChars, fp);
  while (numChars > 0)
  {
    for (charNb = 0; charNb < numChars; charNb++)
    {
      if (buffer[charNb] == '\n')
      {
        printf("\n");
        if (charNb < numChars - 1)
        {
          printf("    ");
        }
      }
      else
      {
        printf("%c", buffer[charNb]);
      }
    }
    numChars = fread(buffer, 1, maxChars, fp);
  }

  if (ferror(fp))
  {
    fclose(fp);
    return 1;
  }
  else
  {
    fclose(fp);
  }

  return 0;
} /* ExportedDataDisplay */

int NewTableDisplay(void)
{
  struct sqlca sqlca;

  printf("\n  SELECT * FROM newtable\n");
  printf("    DEPTNUMB DEPTNAME      \n");
  printf("    -------- --------------\n");

  strcpy(strStmt, "SELECT * FROM newtable");

  EXEC SQL PREPARE stmt FROM :strStmt;
  EMB_SQL_CHECK("statement -- prepare");

  EXEC SQL DECLARE c0 CURSOR FOR stmt;

  EXEC SQL OPEN c0;
  EMB_SQL_CHECK("cursor -- open");

  EXEC SQL FETCH c0 INTO :deptnumb, :deptname;
  EMB_SQL_CHECK("cursor -- fetch");

  while (sqlca.sqlcode != 100)
  {
    printf("    %8d %-s\n", deptnumb, deptname);

    EXEC SQL FETCH c0 INTO :deptnumb, :deptname;
    EMB_SQL_CHECK("cursor -- fetch");
  }

  EXEC SQL CLOSE c0;

  return 0;
} /* NewTableDisplay */

int DataExport(char *dataFileName)
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqldcol dataDescriptor;
  char actionString[256];
  struct sqllob *pAction;
  char msgFileName[128];
  struct db2ExportOut outputInfo;
  struct db2ExportStruct exportParmStruct;

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  db2Export -- Export\n");
  printf("TO EXPORT DATA TO A FILE.\n");

  printf("\n  Be sure to complete all table operations and release\n");
  printf("  all locks before starting an export operation. This\n");
  printf("  can be done by issuing a COMMIT after closing all\n");
  printf("  cursors opened WITH HOLD, or by issuing a ROLLBACK.\n");
  printf("  Please refer to the 'Administrative API Reference'\n");
  printf("  for the details.\n");

  /* export data */
  dataDescriptor.dcolmeth = SQL_METH_D;
  strcpy(actionString, "SELECT deptnumb, deptname FROM org");
  pAction = (struct sqllob *)malloc(sizeof(sqluint32) +
                                    sizeof(actionString) + 1);
  pAction->length = strlen(actionString);
  strcpy(pAction->data, actionString);
  strcpy(msgFileName, "tbexport.MSG");

  exportParmStruct.piDataFileName    = dataFileName;
  exportParmStruct.piLobPathList     = NULL;
  exportParmStruct.piLobFileList     = NULL;
  exportParmStruct.piDataDescriptor  = &dataDescriptor;
  exportParmStruct.piActionString    = pAction;
  exportParmStruct.piFileType        = SQL_DEL;
  exportParmStruct.piFileTypeMod     = NULL;
  exportParmStruct.piMsgFileName     = msgFileName;
  exportParmStruct.iCallerAction     = SQLU_INITIAL;
  exportParmStruct.poExportInfoOut   = &outputInfo;

  printf("\n  Export data.\n");
  printf("    client destination file name: %s\n", dataFileName);
  printf("    action                      : %s\n", actionString);
  printf("    client message file name    : %s\n", msgFileName);

  /* export data */
  db2Export(db2Version820,
            &exportParmStruct,
            &sqlca);

  DB2_API_CHECK("data -- export");

  /* free memory allocated */
  free(pAction);

  /* display exported data */
  rc = ExportedDataDisplay(dataFileName);

  return 0;
} /* DataExport */

int TbImport(char *dataFileName)
{
  int rc = 0;
  struct sqlca sqlca;
  struct sqldcol dataDescriptor;
  char actionString[256];
  struct sqlchar *pAction;
  char msgFileName[128];
  struct db2ImportIn inputInfo;
  struct db2ImportOut outputInfo;
  struct db2ImportStruct importParmStruct;
  long commitcount = 10;

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  db2Import -- Import\n");
  printf("TO IMPORT DATA TO A TABLE.\n");

  /* create new table */
  printf("\n  CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,");
  printf("\n                        deptname VARCHAR(14))\n");

  EXEC SQL CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,
                                 deptname VARCHAR(14));
  EMB_SQL_CHECK("new table -- create");

  /* import table */
  dataDescriptor.dcolmeth = SQL_METH_D;
  strcpy(actionString, "INSERT INTO newtable");
  pAction = (struct sqlchar *)malloc(sizeof(short) +
                                     sizeof(actionString) + 1);
  pAction->length = strlen(actionString);
  strcpy(pAction->data, actionString);
  strcpy(msgFileName, "tbimport.MSG");

  /* Setup db2ImportIn structure */
  inputInfo.iRowcount = inputInfo.iRestartcount = 0;
  inputInfo.iSkipcount = inputInfo.iWarningcount = 0;
  inputInfo.iNoTimeout = 0;
  inputInfo.iAccessLevel = SQLU_ALLOW_NO_ACCESS;
  inputInfo.piCommitcount = &commitcount;
  printf("\n  Import table.\n");
  printf("    client source file name : %s\n", dataFileName);
  printf("    action                  : %s\n", actionString);
  printf("    client message file name: %s\n", msgFileName);

  ImportparmStruct.piDataFileName    = dataFileName;
  importParmStruct.piLobPathList     = NULL;
  importParmStruct.piDataDescriptor  = &dataDescriptor;
  importParmStruct.piActionString    = pAction;
  importParmStruct.piFileType        = SQL_DEL;
  importParmStruct.piFileTypeMod     = NULL;
  importParmStruct.piMsgFileName     = msgFileName;
  importParmStruct.piImportInfoIn    = &inputInfo;
  importParmStruct.poImportInfoOut   = &outputInfo;
  importParmStruct.piNullIndicators  = NULL;
  importParmStruct.iCallerAction     = SQLU_INITIAL;

  /* import table */
  db2Import(db2Version820,
            &importParmStruct,
            &sqlca);

  DB2_API_CHECK("table -- import");

  /* free memory allocated */
  free(pAction);

  /* display import info */
  printf("\n  Import info.\n");
  printf("    rows read     : %ld\n", (int)outputInfo.oRowsRead);
  printf("    rows skipped  : %ld\n", (int)outputInfo.oRowsSkipped);
  printf("    rows inserted : %ld\n", (int)outputInfo.oRowsInserted);
  printf("    rows updated  : %ld\n", (int)outputInfo.oRowsUpdated);
  printf("    rows rejected : %ld\n", (int)outputInfo.oRowsRejected);
  printf("    rows committed: %ld\n", (int)outputInfo.oRowsCommitted);

  /* display content of the new table */
  rc = NewTableDisplay();

  /* drop new table */
  printf("\n  DROP TABLE newtable\n");

  EXEC SQL DROP TABLE newtable;
  EMB_SQL_CHECK("new table -- drop");

  return 0;
} /* TbImport */

int TbLoad(char *dataFileName)
{
  int rc = 0;
  struct sqlca sqlca;

  struct db2LoadStruct paramStruct;
  struct db2LoadIn inputInfoStruct;
  struct db2LoadOut outputInfoStruct;

  struct sqlu_media_list mediaList;
  struct sqldcol dataDescriptor;
  char actionString[256];
  struct sqlchar *pAction;
  char localMsgFileName[128];

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  sqluvqdp -- Quiesce Table Spaces for Table\n");
  printf("  db2Load -- Load\n");
  printf("TO LOAD DATA INTO A TABLE.\n");

  /* create new table */
  printf("\n  CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,");
  printf("\n                        deptname VARCHAR(14))\n");

  EXEC SQL CREATE TABLE newtable(deptnumb SMALLINT NOT NULL,
                                 deptname VARCHAR(14));
  EMB_SQL_CHECK("new table -- create");

  /* quiesce table spaces for table */
  printf("\n  Quiesce the table spaces for 'newtable'.\n");

  EXEC SQL COMMIT;
  EMB_SQL_CHECK("transaction -- commit");

  /* quiesce table spaces for table */
  sqluvqdp("newtable", SQLU_QUIESCEMODE_RESET_OWNED, NULL, &sqlca);
  DB2_API_CHECK("tablespaces for table -- quiesce");

  /* load table */
  mediaList.media_type = SQLU_CLIENT_LOCATION;
  mediaList.sessions = 1;
  mediaList.target.location =
    (struct sqlu_location_entry *)malloc(sizeof(struct sqlu_location_entry) *
                                         mediaList.sessions);
  strcpy(mediaList.target.location->location_entry, dataFileName);

  dataDescriptor.dcolmeth = SQL_METH_D;

  strcpy(actionString, "INSERT INTO newtable");
  pAction = (struct sqlchar *)malloc(sizeof(short) +
                                     sizeof(actionString) + 1);
  pAction->length = strlen(actionString);
  strcpy(pAction->data, actionString);

  strcpy(localMsgFileName, "tbload.MSG");

  /* Setup the input information structure */
  inputInfoStruct.piUseTablespace       = NULL;
  inputInfoStruct.iSavecount    =  0;       /* consistency points  */
                                           /* as infrequently as possible */
  inputInfoStruct.iRestartcount =  0;       /* start at row 1 */
  inputInfoStruct.iRowcount  =  0;       /* load all rows */
  inputInfoStruct.iWarningcount =  0;        /* don't stop for warnings */
  inputInfoStruct.iDataBufferSize =  0;      /* default data buffer size */
  inputInfoStruct.iSortBufferSize  =  0;     /* def. warning buffer size */
  inputInfoStruct.iHoldQuiesce     =  0;     /* don't hold the quiesce */
  inputInfoStruct.iRestartphase    =  ' ';   /* ignored anyway */
  inputInfoStruct.iStatsOpt = SQLU_STATS_NONE; /* don't bother with them */
  inputInfoStruct.iIndexingMode = SQLU_INX_AUTOSELECT;/* let load choose */
                                                      /* indexing mode */
  inputInfoStruct.iCpuParallelism  =  0;
  inputInfoStruct.iNonrecoverable  =  SQLU_NON_RECOVERABLE_LOAD;
  inputInfoStruct.iAccessLevel     =  SQLU_ALLOW_NO_ACCESS;
  inputInfoStruct.iLockWithForce   =  SQLU_NO_FORCE;
  inputInfoStruct.iCheckPending = SQLU_CHECK_PENDING_CASCADE_DEFERRED;

  /* Setup the parameter structure */
  paramStruct.piSourceList = &mediaList;
  paramStruct.piLobPathList = NULL;
  paramStruct.piDataDescriptor = &dataDescriptor;
  paramStruct.piActionString = pAction;
  paramStruct.piFileType = SQL_DEL;
  paramStruct.piFileTypeMod = NULL;
  paramStruct.piLocalMsgFileName = localMsgFileName;
```

```
  paramStruct.piTempFilesPath =   NULL;
  paramStruct.piVendorSortWorkPaths = NULL;
  paramStruct.piCopyTargetList = NULL;
  paramStruct.piNullIndicators = NULL;
  paramStruct.piLoadInfoIn = &inputInfoStruct;
  paramStruct.poLoadInfoOut    = &outputInfoStruct;
  paramStruct.piPartLoadInfoIn  = NULL;
  paramStruct.poPartLoadInfoOut = NULL;
  paramStruct.iCallerAction = SQLU_INITIAL;

  printf("\n  Load table.\n");
  printf("    client source file name : %s\n", dataFileName);
  printf("    action                  : %s\n", actionString);
  printf("    client message file name: %s\n", localMsgFileName);

  /* load table */
  db2Load (db2Version810,            /* Database version number   */
      &paramStruct,             /* In/out parameters         */
      &sqlca);                  /* SQLCA                     */

  DB2_API_CHECK("table -- load");

  /* free memory allocated */
  free(pAction);

  /* display load info */
  printf("\n  Load info.\n");
  printf("    rows read     : %d\n", (int)outputInfoStruct.oRowsRead);
  printf("    rows skipped  : %d\n", (int)outputInfoStruct.oRowsSkipped);
  printf("    rows loaded   : %d\n", (int)outputInfoStruct.oRowsLoaded);
  printf("    rows deleted  : %d\n", (int)outputInfoStruct.oRowsDeleted);
  printf("    rows rejected : %d\n", (int)outputInfoStruct.oRowsRejected);
  printf("    rows committed: %d\n", (int)outputInfoStruct.oRowsCommitted);

  /* display content of the new table */
  rc = NewTableDisplay();

  /* drop new table */
  printf("\n  DROP TABLE newtable\n");

  EXEC SQL DROP TABLE newtable;
  EMB_SQL_CHECK("new table -- drop");

  return 0;
} /* TbLoad */

int TbLoadQuery(void)
{
  int rc = 0;
  struct sqlca sqlca;
  char tableName[128];
  char loadMsgFileName[128];
  db2LoadQueryStruct loadQueryParameters;
  db2LoadQueryOutputStruct loadQueryOutputStructure;

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  db2LoadQuery -- Load Query\n");
  printf("TO CHECK THE STATUS OF A LOAD OPERATION.\n");

  /* Initialize structures */
  memset(&loadQueryParameters, 0, sizeof(db2LoadQueryStruct));
  memset(&loadQueryOutputStructure, 0, sizeof(db2LoadQueryOutputStruct));

  /* Set up the tablename to query. */
  loadQueryParameters.iStringType = DB2LOADQUERY_TABLENAME;
  loadQueryParameters.piString = tableName;
  /* Specify that we want all LOAD messages to be reported. */
  loadQueryParameters.iShowLoadMessages = DB2LOADQUERY_SHOW_ALL_MSGS;

  /* LOAD summary information goes here. */
  loadQueryParameters.poOutputStruct = &loadQueryOutputStructure;

  /* Set up the local message file. */
  loadQueryParameters.piLocalMessageFile = loadMsgFileName;

  /* call the DB2 API */
  strcpy(tableName, "ORG");
  strcpy(loadMsgFileName, "tbldqry.MSG");

  /* load query */
  db2LoadQuery(db2Version810, &loadQueryParameters, &sqlca);
  printf("\n  Note: the table load for '%s' is NOT in progress.\n", tableName);
  printf("  So an empty message file '%s' will be created,\n", loadMsgFileName);
  printf("  and the following values will be zero.\n");
  DB2_API_CHECK("status of load operation -- check");

  printf("\n  Load status has been written to local file %s.\n",
         loadMsgFileName);

  printf("    Number of rows read         = %d\n",
         loadQueryOutputStructure.oRowsRead);

  printf("    Number of rows skipped      = %d\n",
         loadQueryOutputStructure.oRowsSkipped);

  printf("    Number of rows loaded       = %d\n",
         loadQueryOutputStructure.oRowsLoaded);

  printf("    Number of rows rejected     = %d\n",
         loadQueryOutputStructure.oRowsRejected);

  printf("    Number of rows deleted      = %d\n",
         loadQueryOutputStructure.oRowsDeleted);

  printf("    Number of rows committed    = %d\n",
         loadQueryOutputStructure.oRowsCommitted);

  printf("    Number of warnings          = %d\n",
         loadQueryOutputStructure.oWarningCount);

  return 0;
} /* TbLoadQuery */
```

Appendix D. File Formats

Export/Import/Load Utility File Formats

Five operating system file formats supported by the DB2 export, import, and load utilities are described:

DEL
Delimited ASCII, for data exchange among a wide variety of database managers and file managers. This common approach to storing data uses special character delimiters to separate column values.

ASC
Non-delimited ASCII, for importing or loading data from other applications that create flat text files with aligned column data.

PC/IXF
PC version of the Integrated Exchange Format (IXF), the preferred method for data exchange within the database manager. PC/IXF is a structured description of a database table that contains an external representation of the internal table.

WSF
Work-sheet format, for data exchange with products such as Lotus 1-2-3 and Symphony. The load utility does not support this file format.

CURSOR
A cursor declared against an SQL query. This file type is only supported by the load utility.

When using DEL, WSF, or ASC data file formats, define the table, including its column names and data types, before importing the file. The data types in the operating system file fields are converted into the corresponding type of data in the database table. The import utility accepts data with minor incompatibility problems, including character data imported with possible padding or truncation, and numeric data imported into different types of numeric fields.

When using the PC/IXF data file format, the table does not need to exist before beginning the import operation. User-defined distinct types (UDTs) are not made part of the new table column types; instead, the base type is used. Similarly, when exporting to the PC/IXF data file format, UDTs are stored as base data types in the PC/IXF file.

When using the CURSOR file type, the table, including its column names and data types, must be defined before beginning the load operation. The column types of the SQL query must be compatible with the corresponding column types in the target table. It is not necessary for the specified cursor to be open before starting the load operation. The load utility will process the entire result of the query associated with the specified cursor whether or not the cursor has been used to fetch rows.

Related concepts:

- "Queries and table expressions" in *SQL Reference, Volume 1*

Related reference:

- "Delimited ASCII (DEL) File Format" on page 294
- "Non-delimited ASCII (ASC) file format" on page 299

- “PC Version of IXF File Format” on page 302
- “Assignments and comparisons” in *SQL Reference, Volume 1*
- “Casting between data types” in *SQL Reference, Volume 1*

Delimited ASCII (DEL) File Format

A Delimited ASCII (DEL) file is a sequential ASCII file with row and column delimiters. Each DEL file is a stream of ASCII characters consisting of cell values ordered by row, and then by column. Rows in the data stream are separated by row delimiters; within each row, individual cell values are separated by column delimiters.

The following table describes the format of DEL files that can be imported, or that can be generated as the result of an export action.

```
DEL file ::= Row 1 data || Row delimiter ||
             Row 2 data || Row delimiter ||
             .
             .
             .
             Row n data || Optional row delimiter

Row i data ::= Cell value(i,1) || Column delimiter ||
               Cell value(i,2) || Column delimiter ||
               .
               .
               .
               Cell value(i,m)

Row delimiter ::= ASCII line feed sequence[a]

Column delimiter ::= Default value ASCII comma (,)[b]

Cell value(i,j) ::= Leading spaces
                 || ASCII representation of a numeric value
                    (integer, decimal, or float)
                 || Delimited character string
                 || Non-delimited character string
                 || Trailing spaces

Non-delimited character string ::= A set of any characters except a
                                   row delimiter or a column delimiter

Delimited character string ::= A character string delimiter ||
                               An extended character string ||
                               A character string delimiter ||
                               Trailing garbage

Trailing garbage ::= A set of any characters except a row delimiter
                     or a column delimiter

Character string delimiter ::= Default value ASCII double quotation
                               marks (")[c]

extended character string ::= || A set of any characters except a
                                 row delimiter or a character string
                                 delimiter if the NODOUBLEDEL
                                 modifier is specified
                              || A set of any characters except a
                                 row delimiter or a character string
                                 delimiter if the character string
                                 is not part of two consecutive
                                 character string delimiters
                              || A set of any characters except a
```

```
                                      character string delimiter if the
                                      character string delimiter is not
                                      part of two consecutive character
                                      string delimiters, and the DELPRIORITYCHAR
                                      modifier is specified

End-of-file character ::= Hex '1A' (Windows operating system only)

ASCII representation of a numeric value[d] ::= Optional sign '+' or '-'
    || 1 to 31 decimal digits with an optional decimal point before,
       after, or between two digits
    || Optional exponent

Exponent ::= Character 'E' or 'e'
    || Optional sign '+' or '-'
    || 1 to 3 decimal digits with no decimal point

Decimal digit ::= Any one of the characters '0', '1', ... '9'

Decimal point ::= Default value ASCII period (.)[e]
```

- [a] The record delimiter is assumed to be a new line character, ASCII x0A. Data generated on the Windows operating system can use the carriage return/line feed 2-byte standard of 0x0D0A. Data in EBCDIC code pages should use the EBCDIC LF character (0x25) as the record delimiter (EBCDIC data can be loaded using the CODEPAGE option on the LOAD command).
- [b] The column delimiter can be specified with the COLDEL option.
- [c] The character string delimiter can be specified with the CHARDEL option.

  Note: The default priority of delimiters is:
  1. Record delimiter
  2. Character delimiter
  3. Column delimiter
- [d] If the ASCII representation of a numeric value contains an exponent, it is a FLOAT constant. If it has a decimal point but no exponent, it is a DECIMAL constant. If it has no decimal point and no exponent, it is an INTEGER constant.
- [e] The decimal point character can be specified with the DECPT option.

Related reference:
- "DEL Data Type Descriptions" on page 296

Example and Data Type Descriptions

Example DEL File

Following is an example of a DEL file. Each line ends with a line feed sequence (on the Windows operating system, each line ends with a carriage return/line feed sequence).

```
   "Smith, Bob",4973,15.46
   "Jones, Bill",12345,16.34
   "Williams, Sam",452,193.78
```

The following example illustrates the use of non-delimited character strings. The column delimiter has been changed to a semicolon, because the character data contains a comma.

```
Smith, Bob;4973;15.46
Jones, Bill;12345;16.34
Williams, Sam;452;193.78
```

Notes:

1. A space (X'20') is never a valid delimiter.
2. Spaces that precede the first character, or that follow the last character of a cell value, are discarded during import. Spaces that are embedded in a cell value are not discarded.
3. A period (.) is not a valid character string delimiter, because it conflicts with periods in time stamp values.
4. For pure DBCS (graphic), mixed DBCS, and EUC, delimiters are restricted to the range of x00 to x3F, inclusive.
5. For DEL data specified in an EBCDIC code page, the delimiters might not coincide with the shift-in and shift-out DBCS characters.
6. On the Windows operating system, the first occurrence of an end-of-file character (X'1A') that is not within character delimiters indicates the end-of-file. Any subsequent data is not imported.
7. A null value is indicated by the absence of a cell value where one would normally occur, or by a string of spaces.
8. Since some products restrict character fields to 254 or 255 bytes, the export utility generates a warning message whenever a character column of maximum length greater than 254 bytes is selected for export. The import utility accommodates fields that are as long as the longest LONG VARCHAR and LONG VARGRAPHIC columns.

Related reference:

- "Delimited ASCII (DEL) File Format" on page 294
- "DEL Data Type Descriptions" on page 296

DEL Data Type Descriptions

Table 19. Acceptable Data Type Forms for the DEL File Format

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
BIGINT	An INTEGER constant in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807.	ASCII representation of a numeric value in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807. Decimal and float numbers are truncated to integer values.
BLOB, CLOB	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is used as the database column value.
BLOB_FILE, CLOB_FILE	The character data for each BLOB/CLOB column is stored in individual files, and the file name is enclosed by character delimiters.	The delimited or non-delimited name of the file that holds the data.

Table 19. Acceptable Data Type Forms for the DEL File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
CHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is truncated or padded with spaces (X'20'), if necessary, to match the width of the database column.
DATE	*yyyymmdd* (year month day) with no character delimiters. For example: 19931029 Alternatively, the DATESISO option can be used to specify that all date values are to be exported in ISO format.	A delimited or non-delimited character string containing a date value in an ISO format consistent with the territory code of the target database, or a non-delimited character string of the form *yyyymmdd*.
DBCLOB (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is used as the database column value.
DBCLOB_FILE (DBCS only)	The character data for each DBCLOB column is stored in individual files, and the file name is enclosed by character delimiters.	The delimited or non-delimited name of the file that holds the data.
DECIMAL	A DECIMAL constant with the precision and scale of the field being exported. The DECPLUSBLANK option can be used to specify that positive decimal values are to be prefixed with a blank space instead of a plus sign (+).	ASCII representation of a numeric value that does not overflow the range of the database column into which the field is being imported. If the input value has more digits after the decimal point than can be accommodated by the database column, the excess digits are truncated.
FLOAT(long)	A FLOAT constant in the range -10E307 to 10E307.	ASCII representation of a numeric value in the range -10E307 to 10E307.
GRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is truncated or padded with double-byte spaces (for example, X'8140'), if necessary, to match the width of the database column.

Table 19. Acceptable Data Type Forms for the DEL File Format (continued)

Data Type	**Form in Files Created by the Export Utility**	**Form Acceptable to the Import Utility**
INTEGER	An INTEGER constant in the range -2 147 483 648 to 2 147 483 647.	ASCII representation of a numeric value in the range -2 147 483 648 to 2 147 483 647. Decimal and float numbers are truncated to integer values.
LONG VARCHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is used as the database column value.
LONG VARGRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is used as the database column value.
SMALLINT	An INTEGER constant in the range -32 768 to 32 767.	ASCII representation of a numeric value in the range -32 768 to 32 767. Decimal and float numbers are truncated to integer values.
TIME	*hh.mm.ss* (hour minutes seconds). A time value in ISO format enclosed by character delimiters. For example: “09.39.43”	A delimited or non-delimited character string containing a time value in a format consistent with the territory code of the target database.
TIMESTAMP	*yyyy-mm-dd-hh.mm.ss.nnnnnn* (year month day hour minutes seconds microseconds). A character string representing a date and time enclosed by character delimiters.	A delimited or non-delimited character string containing a time stamp value acceptable for storage in a database.
VARCHAR	Character data enclosed by character delimiters (for example, double quotation marks).	A delimited or non-delimited character string. The character string is truncated, if necessary, to match the maximum width of the database column.
VARGRAPHIC (DBCS only)	Graphic data is exported as a delimited character string.	A delimited or non-delimited character string, an even number of bytes in length. The character string is truncated, if necessary, to match the maximum width of the database column.

Related reference:
- “Delimited ASCII (DEL) File Format” on page 294
- “Example DEL File” on page 295
- “Data types” in *SQL Reference, Volume 1*

Non-delimited ASCII (ASC) file format

The non-delimited ASCII format, known as ASC to the import and load utilities, comes in two varieties; fixed length and flexible length. For fixed length ASC, all records are of a fixed length. For flexible length ASC, records are delimited by a row delimiter (always a new line). The term *non-delimited* in non-delimited ASCII means that column values are not separated by delimiters.

When importing or loading ASC data, specifying the RECLEN file type modifier will indicate that the datafile is fixed length ASC. Not specifying it means that the datafile is flexible length ASC.

The non-delimited ASCII format, can be used for data exchange with any ASCII product that has a columnar format for data, including word processors. Each ASC file is a stream of ASCII characters consisting of data values ordered by row and column. Rows in the data stream are separated by row delimiters. Each column within a row is defined by a beginning-ending location pair (specified by IMPORT parameters). Each pair represents locations within a row specified as byte positions. The first position within a row is byte position 1. The first element of each location pair is the byte on which the column begins, and the second element of each location pair is the byte on which the column ends. The columns might overlap. Every row in an ASC file has the same column definition.

An ASC file is defined by:

```
ASC file ::= Row 1 data || Row delimiter ||
             Row 2 data || Row delimiter ||
             .
             .
             .
             Row n data

Row i data ::= ASCII characters || Row delimiter

Row Delimiter ::= ASCII line feed sequence[a]
```

- [a] The record delimiter is assumed to be a new line character, ASCII x0A. Data generated on the Windows operating system can use the carriage return/line feed 2-byte standard of 0x0D0A. Data in EBCDIC code pages should use the EBCDIC LF character (0x25) as the record delimiter (EBCDIC data can be loaded using the CODEPAGE option on the LOAD command). The record delimiter is never interpreted to be part of a field of data.

Related reference:

- "ASC Data Type Descriptions" on page 300

Example and Data Type Descriptions

Example ASC File

Following is an example of an ASC file. Each line ends with a line feed sequence (on the Windows operating system, each line ends with a carriage return/line feed sequence).

```
Smith, Bob       4973     15.46
Jones, Suzanne   12345    16.34
Williams, Sam    452123  193.78
```

Notes:

1. ASC files are assumed not to contain column names.
2. Character strings are *not* enclosed by delimiters. The data type of a column in the ASC file is determined by the data type of the target column in the database table.
3. A NULL is imported into a nullable database column if:
   - A field of blanks is targeted for a numeric, DATE, TIME, or TIMESTAMP database column
   - A field with no beginning and ending location pairs is specified
   - A location pair with beginning and ending locations equal to zero is specified
   - A row of data is too short to contain a valid value for the target column
   - The NULL INDICATORS load option is used, and an `N` (or other value specified by the user) is found in the null indicator column.
4. If the target column is not nullable, an attempt to import a field of blanks into a numeric, DATE, TIME, or TIMESTAMP column causes the row to be rejected.
5. If the input data is not compatible with the target column, and that column is nullable, a null is imported or the row is rejected, depending on where the error is detected. If the column is not nullable, the row is rejected. Messages are written to the message file, specifying incompatibilities that are found.

Related reference:

- "ASC Data Type Descriptions" on page 300

ASC Data Type Descriptions

Table 20. Acceptable Data Type Forms for the ASC File Format

Data Type	Form Acceptable to the Import Utility
BIGINT	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
BLOB/CLOB	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
BLOB_FILE, CLOB_FILE, DBCLOB_FILE (DBCS only)	A delimited or non-delimited name of the file that holds the data.
CHAR	A string of characters. The character string is truncated or padded with spaces on the right, if necessary, to match the width of the target column.

Table 20. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	Form Acceptable to the Import Utility
DATE	A character string representing a date value in a format consistent with the territory code of the target database. The beginning and ending locations should specify a field width that is within the range for the external representation of a date.
DBCLOB (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.
DECIMAL	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range of the database column into which they are being imported. If the input value has more digits after the decimal point than the scale of the database column, the excess digits are truncated. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
FLOAT(long)	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. All values are valid. A comma, period, or colon is considered to be a decimal point. An uppercase or lowercase E is accepted as the beginning of the exponent of a FLOAT constant. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
GRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated or padded with double-byte spaces (0x8140) on the right, if necessary, to match the maximum length of the target column.
INTEGER	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -2 147 483 648 to 2 147 483 647. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
LONG VARCHAR	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.

Table 20. Acceptable Data Type Forms for the ASC File Format (continued)

Data Type	**Form Acceptable to the Import Utility**
LONG VARGRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.
SMALLINT	A constant in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -32 768 to 32 767. Decimal numbers are truncated to integer values. A comma, period, or colon is considered to be a decimal point. Thousands separators are not allowed. The beginning and ending locations should specify a field whose width does not exceed 50 bytes. Integers, decimal numbers, and the mantissas of floating point numbers can have no more than 31 digits. Exponents of floating point numbers can have no more than 3 digits.
TIME	A character string representing a time value in a format consistent with the territory code of the target database. The beginning and ending locations should specify a field width that is within the range for the external representation of a time.
TIMESTAMP	A character string representing a time stamp value acceptable for storage in a database. The beginning and ending locations should specify a field width that is within the range for the external representation of a time stamp.
VARCHAR	A string of characters. The character string is truncated on the right, if necessary, to match the maximum length of the target column. If the ASC *truncate blanks* option is in effect, trailing blanks are stripped from the original or the truncated string.
VARGRAPHIC (DBCS only)	A string of an even number of bytes. A string of an odd number of bytes is invalid and is not accepted. A valid string is truncated on the right, if necessary, to match the maximum length of the target column.

Related reference:

- "Example ASC File" on page 299
- "Data types" in *SQL Reference, Volume 1*

PC Version of IXF File Format

The PC version of IXF (PC/IXF) file format is a database manager adaptation of the Integration Exchange Format (IXF) data interchange architecture. The IXF architecture was specifically designed to enable the exchange of relational database structures and data. The PC/IXF architecture allows the database manager to export a database without having to anticipate the requirements and idiosyncrasies of a receiving product. Similarly, a product importing a PC/IXF file need only understand the PC/IXF architecture; the characteristics of the product which exported the file are not relevant. The PC/IXF file architecture maintains the independence of both the exporting and the importing database systems.

The IXF architecture is a generic relational database exchange format that supports a rich set of relational data types, including some types that might not be supported by specific relational database products. The PC/IXF file format preserves this flexibility; for example, the PC/IXF architecture supports both single-byte character string (SBCS) and double-byte character string (DBCS) data types. Not all implementations support all PC/IXF data types; however, even restricted implementations provide for the detection and disposition of unsupported data types during import.

In general, a PC/IXF file consists of an unbroken sequence of variable-length records. The file contains the following record types in the order shown:
- One header record of record type H
- One table record of record type T
- Multiple column descriptor records of record type C (one record for each column in the table)
- Multiple data records of record type D (each row in the table is represented by one or more D records).

A PC/IXF file might also contain application records of record type A, anywhere after the H record. These records are permitted in PC/IXF files to enable an application to include additional data, not defined by the PC/IXF format, in a PC/IXF file. A records are ignored by any program reading a PC/IXF file that does not have particular knowledge about the data format and content implied by the application identifier in the A record.

Every record in a PC/IXF file begins with a record length indicator. This is a 6-byte right justified character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Programs reading PC/IXF files should use these record lengths to locate the end of the current record and the beginning of the next record. H, T, and C records must be sufficiently large to include all of their defined fields, and, of course, their record length fields must agree with their actual lengths. However, if extra data (for example, a *new* field), is added to the end of one of these records, pre-existing programs reading PC/IXF files should ignore the extra data, and generate no more than a warning message. Programs writing PC/IXF files, however, should write H, T and C records that are the precise length needed to contain all of the defined fields.

If a PC/IXF file contains LOB Location Specifier (LLS) columns, each LLS column must have its own D record. D records are automatically created by the export utility, but you will need to create them manually if you are using a third party tool to generate the PC/IXF files. Further, an LLS is required for each LOB column in a table, including those with a null value. If a LOB column is null, you will need to create an LLS representing a null LOB.

The D record entry for each XML column will contain two bytes little endian indicating the XML data specifier (XDS) length, followed by the XDS itself.

For example, the following XDS:

```
<XDS FIL="a.xml" OFF="1000" LEN="100" SCH="RENATA.SCHEMA" />
```

will be represented by the following bytes in a D record:

```
0x3D 0x00 <XDS FIL="a.xml" OFF="1000" LEN="100" SCH="RENATA.SCHEMA" />
```

PC/IXF file records are composed of fields which contain character data. The import and export utilities interpret this character data using the CPGID of the target database, with two exceptions:

- The IXFADATA field of A records.

  The code page environment of character data contained in an IXFADATA field is established by the application which creates and processes a particular A record; that is, the environment varies by implementation.
- The IXFDCOLS field of D records.

  The code page environment of character data contained in an IXFDCOLS field is a function of information contained in the C record which defines a particular column and its data.

Numeric fields in H, T, and C records, and in the prefix portion of D and A records should be right justified single-byte character representations of integer values, filled with leading zeros or blanks. A value of zero should be indicated with at least one (right justified) zero character, not blanks. Whenever one of these numeric fields is not used, for example IXFCLENG, where the length is implied by the data type, it should be filled with blanks. These numeric fields are:

```
IXFHRECL, IXFTRECL, IXFCRECL, IXFDRECL, IXFARECL,
IXFHHCNT, IXFHSBCP, IXFHDBCP, IXFTCCNT, IXFTNAML,
IXFCLENG, IXFCDRID, IXFCPOSN, IXFCNAML, IXFCTYPE,
IXFCSBCP, IXFCDBCP, IXFCNDIM, IXFCDSIZ, IXFDRID
```

Note: The database manager PC/IXF file format is not identical to the System/370.

Related reference:

- "Data Type-Specific Rules Governing PC/IXF File Import into Databases" on page 330
- "Differences Between PC/IXF and Version 0 System/370 IXF" on page 338
- "FORCEIN Option" on page 332
- "General Rules Governing PC/IXF File Import into Databases" on page 328
- "PC/IXF Data Type Descriptions" on page 325
- "PC/IXF data types" on page 320
- "PC/IXF Record Types" on page 304

PC Version of IXF File Format - Details

PC/IXF Record Types

There are five basic PC/IXF record types:

- header
- table
- column descriptor
- data
- application

and six application subtypes that IBM DB2 V9.1 uses:

- index
- hierarchy
- subtable

- continuation
- terminate
- identity

Each PC/IXF record type is defined as a sequence of fields; these fields are required, and must appear in the order shown.

```
HEADER RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFHRECL       06-BYTE   CHARACTER   record length
   IXFHRECT       01-BYTE   CHARACTER   record type = 'H'
   IXFHID         03-BYTE   CHARACTER   IXF identifier
   IXFHVERS       04-BYTE   CHARACTER   IXF version
   IXFHPROD       12-BYTE   CHARACTER   product
   IXFHDATE       08-BYTE   CHARACTER   date written
   IXFHTIME       06-BYTE   CHARACTER   time written
   IXFHHCNT       05-BYTE   CHARACTER   heading record count
   IXFHSBCP       05-BYTE   CHARACTER   single byte code page
   IXFHDBCP       05-BYTE   CHARACTER   double byte code page
   IXFHFIL1       02-BYTE   CHARACTER   reserved
```

The following fields are contained in the header record:

IXFHRECL
: The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The H record must be sufficiently long to include all of its defined fields.

IXFHRECT
: The IXF record type, which is set to H for this record.

IXFHID
: The file format identifier, which is set to IXF for this file.

IXFHVERS
: The PC/IXF format level used when the file was created, which is set to ’0002’.

IXFHPROD
: A field that can be used by the program creating the file to identify itself. If this field is filled in, the first six bytes are used to identify the product creating the file, and the last six bytes are used to indicate the version or release of the creating product. The database manager uses this field to signal the existence of database manager-specific data.

IXFHDATE
: The date on which the file was written, in the form *yyyymmdd*.

IXFHTIME
: The time at which the file was written, in the form *hhmmss*. This field is optional and can be left blank.

IXFHHCNT
: The number of H, T, and C records in this file that precede the first data record. A records are not included in this count.

IXFHSBCP
: Single-byte code page field, containing a single-byte character representation of a SBCS CPGID or ’00000’.

The export utility sets this field equal to the SBCS CPGID of the exported database table. For example, if the table SBCS CPGID is 850, this field contains ’00850’.

IXFHDBCP
Double-byte code page field, containing a single-byte character representation of a DBCS CPGID or ’00000’.

The export utility sets this field equal to the DBCS CPGID of the exported database table. For example, if the table DBCS CPGID is 301, this field contains ’00301’.

IXFHFIL1
Spare field set to two blanks to match a reserved field in host IXF files.

```
TABLE RECORD

   FIELD NAME     LENGTH     TYPE       COMMENTS
   ----------     -------    ---------  -------------

   IXFTRECL       006-BYTE   CHARACTER  record length
   IXFTRECT       001-BYTE   CHARACTER  record type = 'T'
   IXFTNAML       003-BYTE   CHARACTER  name length
   IXFTNAME       256-BYTE   CHARACTER  name of data
   IXFTQULL       003-BYTE   CHARACTER  qualifier length
   IXFTQUAL       256-BYTE   CHARACTER  qualifier
   IXFTSRC        012-BYTE   CHARACTER  data source
   IXFTDATA       001-BYTE   CHARACTER  data convention = 'C'
   IXFTFORM       001-BYTE   CHARACTER  data format = 'M'
   IXFTMFRM       005-BYTE   CHARACTER  machine format = 'PC'
   IXFTLOC        001-BYTE   CHARACTER  data location = 'I'
   IXFTCCNT       005-BYTE   CHARACTER  'C' record count
   IXFTFIL1       002-BYTE   CHARACTER  reserved
   IXFTDESC       030-BYTE   CHARACTER  data description
   IXFTPKNM       257-BYTE   CHARACTER  primary key name
   IXFTDSPC       257-BYTE   CHARACTER  reserved
   IXFTISPC       257-BYTE   CHARACTER  reserved
   IXFTLSPC       257-BYTE   CHARACTER  reserved
```

The following fields are contained in the table record:

IXFTRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The T record must be sufficiently long to include all of its defined fields.

IXFTRECT
The IXF record type, which is set to T for this record.

IXFTNAML
The length, in bytes, of the table name in the IXFTNAME field.

IXFTNAME
The name of the table. If each file has only one table, this is an informational field only. The database manager does not use this field when importing data. When writing a PC/IXF file, the database manager writes the DOS file name (and possibly path information) to this field.

IXFTQULL
The length, in bytes, of the table name qualifier in the IXFTQUAL field.

IXFTQUAL
Table name qualifier, which identifies the creator of a table in a relational

system. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFTSRC
Used to indicate the original source of the data. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFTDATA
Convention used to describe the data. This field must be set to `C` for import and export, indicating that individual column attributes are described in the following column descriptor (C) records, and that data follows PC/IXF conventions.

IXFTFORM
Convention used to store numeric data. This field must be set to `M`, indicating that numeric data in the data (D) records is stored in the machine (internal) format specified by the IXFTMFRM field.

IXFTMFRM
The format of any machine data in the PC/IXF file. The database manager will only read or write files if this field is set to `PC`*bbb*, where *b* represents a blank, and PC specifies that data in the PC/IXF file is in IBM PC machine format.

IXFTLOC
The location of the data. The database manager only supports a value of I, meaning the data is internal to this file.

IXFTCCNT
The number of C records in this table. It is a right-justified character representation of an integer value.

IXFTFIL1
Spare field set to two blanks to match a reserved field in host IXF files.

IXFTDESC
Descriptive data about the table. This is an informational field only. If a program writing a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field. This field contains `NOT NULL WITH DEFAULT` if the column was not null with default, and the table name came from a workstation database.

IXFTPKNM
The name of the primary key defined on the table (if any). The name is stored as a null-terminated string.

IXFTDSPC
This field is reserved for future use.

IXFTISPC
This field is reserved for future use.

IXFTLSPC
This field is reserved for future use.

```
COLUMN DESCRIPTOR RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     -------    ---------   -------------
   IXFCRECL       006-BYTE   CHARACTER   record length
   IXFCRECT       001-BYTE   CHARACTER   record type = 'C'
   IXFCNAML       003-BYTE   CHARACTER   column name length
   IXFCNAME       256-BYTE   CHARACTER   column name
   IXFCNULL       001-BYTE   CHARACTER   column allows nulls
   IXFCDEF        001-BYTE   CHARACTER   column has defaults
   IXFCSLCT       001-BYTE   CHARACTER   column selected flag
   IXFCKPOS       002-BYTE   CHARACTER   position in primary key
   IXFCCLAS       001-BYTE   CHARACTER   data class
   IXFCTYPE       003-BYTE   CHARACTER   data type
   IXFCSBCP       005-BYTE   CHARACTER   single byte code page
   IXFCDBCP       005-BYTE   CHARACTER   double byte code page
   IXFCLENG       005-BYTE   CHARACTER   column data length
   IXFCDRID       003-BYTE   CHARACTER   'D' record identifier
   IXFCPOSN       006-BYTE   CHARACTER   column position
   IXFCDESC       030-BYTE   CHARACTER   column description
   IXFCLOBL       020-BYTE   CHARACTER   lob column length
   IXFCUDTL       003-BYTE   CHARACTER   UDT name length
   IXFCUDTN       256-BYTE   CHARACTER   UDT name
   IXFCDEFL       003-BYTE   CHARACTER   default value length
   IXFCDEFV       254-BYTE   CHARACTER   default value
   IXFCDLPR       010-BYTE   CHARACTER   datalink properties
   IXFCREF        001-BYTE   CHARACTER   reference type
   IXFCNDIM       002-BYTE   CHARACTER   number of dimensions
   IXFCDSIZ       varying    CHARACTER   size of each dimension
```

The following fields are contained in column descriptor records:

IXFCRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. The C record must be sufficiently long to include all of its defined fields.

IXFCRECT
The IXF record type, which is set to C for this record.

IXFCNAML
The length, in bytes, of the column name in the IXFCNAME field.

IXFCNAME
The name of the column.

IXFCNULL
Specifies if nulls are permitted in this column. Valid settings are `Y` or `N`.

IXFCDEF
Specifies if a default value is defined for this field. Valid settings are `Y` or `N`.

IXFCSLCT
An obsolete field whose intended purpose was to allow selection of a subset of columns in the data. Programs writing PC/IXF files should always store a Y in this field. Programs reading PC/IXF files should ignore the field.

IXFCKPOS
The position of the column as part of the primary key. Valid values range from `01` to `16`, or `N` if the column is not part of the primary key.

IXFCCLAS
The class of data types to be used in the IXFCTYPE field. The database manager only supports relational types (R).

IXFCTYPE
The data type for the column.

IXFCSBCP
Contains a single-byte character representation of a SBCS CPGID. This field specifies the CPGID for single-byte character data, which occurs with the IXFDCOLS field of the D records for this column.

The semantics of this field vary with the data type for the column (specified in the IXFCTYPE field).

- For a character string column, this field should normally contain a non-zero value equal to that of the IXFHSBCP field in the H record; however, other values are permitted. If this value is zero, the column is interpreted to contain bit string data.
- For a numeric column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a date or time column, this field is not meaningful. It is set to the value of the IXFHSBCP field by the export utility, and ignored by the import utility.
- For a graphic column, this field must be zero.

IXFCDBCP
Contains a single-byte character representation of a DBCS CPGID. This field specifies the CPGID for double-byte character data, which occurs with the IXFDCOLS field of the D records for this column.

The semantics of this field vary with the data type for the column (specified in the IXFCTYPE field).

- For a character string column, this field should either be zero, or contain a value equal to that of the IXFHDBCP field in the H record; however, other values are permitted. If the value in the IXFCSBCP field is zero, the value in this field must be zero.
- For a numeric column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a date or time column, this field is not meaningful. It is set to zero by the export utility, and ignored by the import utility.
- For a graphic column, this field must have a value equal to the value of the IXFHDBCP field.

IXFCLENG
Provides information about the size of the column being described. For some data types, this field is unused, and should contain blanks. For other data types, this field contains the right-justified character representation of an integer specifying the column length. For yet other data types, this field is divided into two subfields: 3 bytes for precision, and 2 bytes for scale; both of these subfields are right-justified character representations of integers.

IXFCDRID
The D record identifier. This field contains the right-justified character representation of an integer value. Several D records can be used to contain each row of data in the PC/IXF file. This field specifies which D record (of the several D records contributing to a row of data) contains the data for

the column. A value of one (for example, 001) indicates that the data for a column is in the first D record in a row of data. The first C record must have an IXFCDRID value of one. All subsequent C records must have an IXFCDRID value equal to the value in the preceding C record, or one higher.

IXFCPOSN
The value in this field is used to locate the data for the column within one of the D records representing a row of table data. It is the starting position of the data for this column within the IXFDCOLS field of the D record. If the column is nullable, IXFCPOSN points to the null indicator; otherwise, it points to the data itself. If a column contains varying length data, the data itself begins with the current length indicator. The IXFCPOSN value for the first byte in the IXFDCOLS field of the D record is one (not zero). If a column is in a new D record, the value of IXFCPOSN should be one; otherwise, IXFCPOSN values should increase from column to column to such a degree that the data values do not overlap.

IXFCDESC
Descriptive information about the column. This is an informational field only. If a program writing to a file has no data to write to this field, the preferred fill value is blanks. Programs reading a file might print or display this field, or store it in an informational field, but no computations should depend on the content of this field.

IXFCLOBL
The length, in bytes, of the long or the LOB defined in this column. If this column is not a long or a LOB, the value in this field is 000.

IXFCUDTL
The length, in bytes, of the user defined type (UDT) name in the IXFCUDTN field. If the type of this column is not a UDT, the value in this field is 000.

IXFCUDTN
The name of the user defined type that is used as the data type for this column.

IXFCDEFL
The length, in bytes, of the default value in the IXFCDEFV field. If this column does not have a default value, the value in this field is 000.

IXFCDEFV
Specifies the default value for this column, if one has been defined.

IXFCDLPR
If the column is a DATALINK column, this field describes the following properties:

- The first character represents the link type, and has a value of U.
- The second character represents the link control type. Valid values are N for no control, and F for file control.
- The third character represents the level of integrity, and has a value of A (for database manager controlling all DATALINK values).
- The fourth character represents read permission. Valid values are D for database determined permissions, and F for file system determined permissions.
- The fifth character represents write permission. Valid values are B for blocked access, and F for file system determined permissions.

- The sixth character represents recovery options. Valid values are Y (DB2 will support point-in-time recovery of files referenced in this column), and N (no support).
- The seventh character represents the action that is to be taken when the data file is unlinked. Valid values are R for restore, and D for delete the file.

IXFCREF
If the column is part of a hierarchy, this field specifies whether the column is a data column (D), or a reference column (R).

IXFCNDIM
The number of dimensions in the column. Arrays are not supported in this version of PC/IXF. This field must therefore contain a character representation of a zero integer value.

IXFCDSIZ
The size or range of each dimension. The length of this field is five bytes per dimension. Since arrays are not supported (that is, the number of dimensions must be zero), this field has zero length, and does not actually exist.

```
DATA RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFDRECL       06-BYTE   CHARACTER   record length
   IXFDRECT       01-BYTE   CHARACTER   record type = 'D'
   IXFDRID        03-BYTE   CHARACTER   'D' record identifier
   IXFDFIL1       04-BYTE   CHARACTER   reserved
   IXFDCOLS       varying   variable    columnar data
```

The following fields are contained in the data records:

IXFDRECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each D record must be sufficiently long to include all significant data for the current occurrence of the last data column stored in the record.

IXFDRECT
The IXF record type, which is set to D for this record, indicating that it contains data values for the table.

IXFDRID
The record identifier, which identifies a particular D record within the sequence of several D records contributing to a row of data. For the first D record in a row of data, this field has a value of one; for the second D record in a row of data, this field has a value of two, and so on. In each row of data, all the D record identifiers called out in the C records must actually exist.

IXFDFIL1
Spare field set to four blanks to match reserved fields, and hold a place for a possible shift-out character, in host IXF files.

IXFDCOLS
The area for columnar data. The data area of a data record (D record) is composed of one or more column entries. There is one column entry for each column descriptor record, which has the same D record identifier as

the D record. In the D record, the starting position of the column entries is indicated by the IXFCPOSN value in the C records.

The format of the column entry data depends on whether or not the column is nullable:

- If the column is nullable (the IXFCNULL field is set to `Y`), the column entry data includes a null indicator. If the column is not null, the indicator is followed by data type-specific information, including the actual database value. The null indicator is a two-byte value set to x'0000' for not null, and x'FFFF' for null.
- If the column is not nullable, the column entry data includes only data type-specific information, including the actual database value.

For varying-length data types, the data type-specific information includes a current length indicator. The current length indicators are 2-byte integers in a form specified by the IXFTMFRM field.

The length of the data area of a D record cannot exceed 32 771 bytes.

```
APPLICATION RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFARECL       06-BYTE   CHARACTER   record length
   IXFARECT       01-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       12-BYTE   CHARACTER   application identifier
   IXFADATA       varying   variable    application-specific data
```

The following fields are contained in application records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies the application creating the A record. PC/IXF files created by the database manager can have A records with the first 6 characters of this field set to a constant identifying the database manager, and the last 6 characters identifying the release or version of the database manager or another application writing the A record.

IXFADATA
This field contains application dependent supplemental data, whose form and content are known only to the program creating the A record, and to other applications which are likely to process the A record.

```
DB2 INDEX RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
```

```
IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                        'DB2    02.00'
IXFAITYP       001-BYTE   CHARACTER   application specific data type =
                                        'I'
IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
IXFANDXL       002-BYTE   SHORT INT   length of name of the index
IXFANDXN       256-BYTE   CHARACTER   name of the index
IXFANCL        002-BYTE   SHORT INT   length of name of the index creator
IXFANCN        256-BYTE   CHARACTER   name of the index creator
IXFATABL       002-BYTE   SHORT INT   length of name of the table
IXFATABN       256-BYTE   CHARACTER   name of the table
IXFATCL        002-BYTE   SHORT INT   length of name of the table creator
IXFATCN        256-BYTE   CHARACTER   name of the table creator
IXFAUNIQ       001-BYTE   CHARACTER   unique rule
IXFACCNT       002-BYTE   CHARACTER   column count
IXFAREVS       001-BYTE   CHARACTER   allow reverse scan flag
IXFAPCTF       002-BYTE   CHARACTER   amount of pct free
IXFAPCTU       002-BYTE   CHARACTER   amount of minpctused
IXFAEXTI       001-BYTE   CHARACTER   reserved
IXFACNML       002-BYTE   SHORT INT   length of name of the columns
IXFACOLN       varying    CHARACTER   name of the columns in the index
```

One record of this type is specified for each user defined index. This record is located after all of the C records for the table. The following fields are contained in DB2 index records:

IXFARECL
: The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
: The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
: The application identifier, which identifies DB2 as the application creating this A record.

IXFAITYP
: Specifies that this is subtype ″I″ of DB2 application records.

IXFADATE
: The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
: The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFANDXL
: The length, in bytes, of the index name in the IXFANDXN field.

IXFANDXN
: The name of the index.

IXFANCL
: The length, in bytes, of the index creator name in the IXFANCN field.

IXFANCN
: The name of the index creator.

IXFATABL
: The length, in bytes, of the table name in the IXFATABN field.

IXFATABN
: The name of the table.

IXFATCL
: The length, in bytes, of the table creator name in the IXFATCN field.

IXFATCN
: The name of the table creator.

IXFAUNIQ
: Specifies the type of index. Valid values are `P` for a primary key, `U` for a unique index, and `D` for a non unique index.

IXFACCNT
: Specifies the number of columns in the index definition.

IXFAREVS
: Specifies whether reverse scan is allowed on this index. Valid values are `Y` for reverse scan, and `N` for no reverse scan.

IXFAPCTF
: Specifies the percentage of index pages to leave as free. Valid values range from `-1` to `99`. If a value of `-1` or zero is specified, the system default value is used.

IXFAPCTU
: Specifies the minimum percentage of index pages that must be free before two index pages can be merged. Valid values range from `00` to `99`.

IXFAEXTI
: Reserved for future use.

IXFACNML
: The length, in bytes, of the column names in the IXFACOLN field.

IXFACOLN
: The names of the columns that are part of this index. Valid values are in the form `+`*`name`*`-`*`name`*`...`, where `+` specifies an ascending sort on the column, and `-` specifies a descending sort on the column.

```
DB2 HIERARCHY RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                           'DB2    02.00'
   IXFAXTYP       001-BYTE   CHARACTER   application specific data type =
                                           'X'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
   IXFAYCNT       010-BYTE   CHARACTER   'Y' record count for this hierarchy
   IXFAYSTR       010-BYTE   CHARACTER   starting column of this hierarchy
```

One record of this type is used to describe a hierarchy. All subtable records (see below) must be located immediately after the hierarchy record, and hierarchy records are located after all of the C records for the table. The following fields are contained in DB2 hierarchy records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFAXTYP
Specifies that this is subtype ″X″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFAYCNT
Specifies the number of subtable records that are expected after this hierarchy record.

IXFAYSTR
Specifies the index of the subtable records at the beginning of the exported data. If export of a hierarchy was started from a non-root subtable, all parent tables of this subtable are exported. The position of this subtable inside of the IXF file is also stored in this field. The first X record represents the column with an index of zero.

```
DB2 SUBTABLE RECORD

   FIELD NAME     LENGTH     TYPE       COMMENTS
   ----------     --------   ---------  -------------
   IXFARECL       006-BYTE   CHARACTER  record length
   IXFARECT       001-BYTE   CHARACTER  record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER  application identifier =
                                          'DB2    02.00'
   IXFAYTYP       001-BYTE   CHARACTER  application specific data type =
                                          'Y'
   IXFADATE       008-BYTE   CHARACTER  date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER  time written from the 'H' record
   IXFASCHL       003-BYTE   CHARACTER  type schema name length
   IXFASCHN       256-BYTE   CHARACTER  type schema name
   IXFATYPL       003-BYTE   CHARACTER  type name length
   IXFATYPN       256-BYTE   CHARACTER  type name
   IXFATABL       003-BYTE   CHARACTER  table name length
   IXFATABN       256-BYTE   CHARACTER  table name
   IXFAPNDX       010-BYTE   CHARACTER  subtable index of parent table
   IXFASNDX       005-BYTE   CHARACTER  starting column index of current
                                          table
   IXFAENDX       005-BYTE   CHARACTER  ending column index of current
                                          table
```

One record of this type is used to describe a subtable as part of a hierarchy. All subtable records belonging to a hierarchy must be stored together, and immediately after the corresponding hierarchy record. A subtable is composed of one or more columns, and each column is described in a column record. Each column in a subtable must be described in a consecutive set of C records. The following fields are contained in DB2 subtable records:

IXFARECL
: The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
: The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
: The application identifier, which identifies DB2 as the application creating this A record.

IXFAYTYP
: Specifies that this is subtype ″Y″ of DB2 application records.

IXFADATE
: The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
: The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFASCHL
: The length, in bytes, of the subtable schema name in the IXFASCHN field.

IXFASCHN
: The name of the subtable schema.

IXFATYPL
: The length, in bytes, of the subtable name in the IXFATYPN field.

IXFATYPN
: The name of the subtable.

IXFATABL
: The length, in bytes, of the table name in the IXFATABN field.

IXFATABN
: The name of the table.

IXFAPNDX
: Subtable record index of the parent subtable. If this subtable is the root of a hierarchy, this field contains the value -1.

IXFASNDX
: Starting index of the column records that made up this subtable.

IXFAENDX
: Ending index of the column records that made up this subtable.

```
DB2 CONTINUATION RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                           'DB2    02.00'
   IXFACTYP       001-BYTE   CHARACTER   application specific data type = 'C'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
   IXFALAST       002-BYTE   SHORT INT   last diskette volume number
   IXFATHIS       002-BYTE   SHORT INT   this diskette volume number
   IXFANEXT       002-BYTE   SHORT INT   next diskette volume number
```

This record is found at the end of each file that is part of a multi-volume IXF file, unless that file is the final volume; it can also be found at the beginning of each file that is part of a multi-volume IXF file, unless that file is the first volume. The purpose of this record is to keep track of file order. The following fields are contained in DB2 continuation records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFACTYP
Specifies that this is subtype ″C″ of DB2 application records.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFALAST
This field is a binary field, in little-endian format. The value should be one less than the value in IXFATHIS.

IXFATHIS
This field is a binary field, in little-endian format. The value in this field on consecutive volumes should also be consecutive. The first volume has a value of 1.

IXFANEXT
This field is a binary field, in little-endian format. The value should be one more than the value in IXFATHIS, unless the record is at the beginning of the file, in which case the value should be zero.

```
DB2 TERMINATE RECORD

   FIELD NAME     LENGTH     TYPE        COMMENTS
   ----------     --------   ---------   -------------
   IXFARECL       006-BYTE   CHARACTER   record length
   IXFARECT       001-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       012-BYTE   CHARACTER   application identifier =
                                           'DB2    02.00'
   IXFAETYP       001-BYTE   CHARACTER   application specific data type =
                                           'E'
   IXFADATE       008-BYTE   CHARACTER   date written from the 'H' record
   IXFATIME       006-BYTE   CHARACTER   time written from the 'H' record
```

This record is the end-of-file marker found at the end of an IXF file. The following fields are contained in DB2 terminate records:

IXFARECL
: The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
: The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
: The application identifier, which identifies DB2 as the application creating this A record.

IXFAETYP
: Specifies that this is subtype ″E″ of DB2 application records.

IXFADATE
: The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
: The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

```
DB2 IDENTITY RECORD

   FIELD NAME     LENGTH    TYPE        COMMENTS
   ----------     -------   ---------   -------------
   IXFARECL       06-BYTE   CHARACTER   record length
   IXFARECT       01-BYTE   CHARACTER   record type = 'A'
   IXFAPPID       12-BYTE   CHARACTER   application identifier
   IXFATYPE       01-BYTE   CHARACTER   application specific record type = 'S'
   IXFADATE       08-BYTE   CHARACTER   application record creation date
   IXFATIME       06-BYTE   CHARACTER   application record creation time
   IXFACOLN       06-BYTE   CHARACTER   column number of the identity column
   IXFAITYP       01-BYTE   CHARACTER   generated always ('Y' or 'N')
   IXFASTRT       33-BYTE   CHARACTER   identity START AT value
   IXFAINCR       33-BYTE   CHARACTER   identity INCREMENT BY value
   IXFACACH       10-BYTE   CHARACTER   identity CACHE value
   IXFAMINV       33-BYTE   CHARACTER   identity MINVALUE
   IXFAMAXV       33-BYTE   CHARACTER   identity MAXVALUE
   IXFACYCL       01-BYTE   CHARACTER   identity CYCLE ('Y' or 'N')
   IXFAORDR       01-BYTE   CHARACTER   identity ORDER ('Y' or 'N')
   IXFARMRL       03-BYTE   CHARACTER   identity Remark length
   IXFARMRK      254-BYTE   CHARACTER   identity Remark value
```

The following fields are contained in DB2 identity records:

IXFARECL
The record length indicator. A 6-byte character representation of an integer value specifying the length, in bytes, of the portion of the PC/IXF record that follows the record length indicator; that is, the total record size minus 6 bytes. Each A record must be sufficiently long to include at least the entire IXFAPPID field.

IXFARECT
The IXF record type, which is set to A for this record, indicating that this is an application record. These records are ignored by programs which do not have particular knowledge about the content and the format of the data implied by the application identifier.

IXFAPPID
The application identifier, which identifies DB2 as the application creating this A record.

IXFATYPE
Application specific record type. This field should always have a value of ″S″.

IXFADATE
The date on which the file was written, in the form *yyyymmdd*. This field must have the same value as IXFHDATE.

IXFATIME
The time at which the file was written, in the form *hhmmss*. This field must have the same value as IXFHTIME.

IXFACOLN
Column number of the identity column in the table.

IXFAITYP
The type of the identity column. A value of ″Y″ indicates that the identity column is always GENERATED. All other values are interpreted to mean that the column is of type GENERATED BY DEFAULT.

IXFASTRT
The START AT value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFAINCR
The INCREMENT BY value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFACACH
The CACHE value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation. A value of ″1″ corresponds to the NO CACHE option.

IXFAMINV
The MINVALUE for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFAMAXV
The MAXVALUE for the identity column that was supplied to the CREATE TABLE statement at the time of table creation.

IXFACYCL
The CYCLE value for the identity column that was supplied to the

CREATE TABLE statement at the time of table creation. A value of ″Y″ corresponds to the CYCLE option, any other value corresponds to NO CYCLE.

IXFAORDR
The ORDER value for the identity column that was supplied to the CREATE TABLE statement at the time of table creation. A value of ″Y″ corresponds to the ORDER option, any other value corresponds to NO ORDER.

IXFARMRL
The length, in bytes, of the remark in IXFARMRK field.

IXFARMRK
This is the user-entered remark associated with the identity column. This is an informational field only. The database manager does not use this field when importing data.

Related reference:

- "PC/IXF Data Type Descriptions" on page 325
- "PC/IXF data types" on page 320

PC/IXF data types

Table 21. PC/IXF Data Types

Name	IXFCTYPE Value	Description
BIGINT	492	An 8-byte integer in the form specified by IXFTMFRM. It represents a whole number between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807. IXFCSBCP and IXFCDBCP are not significant , and should be zero. IXFCLENG is not used, and should contain blanks.
BLOB, CLOB	404, 408	A variable-length character string. The maximum length of the string is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 32 767 bytes. The string itself is preceded by a current length indicator, which is a 4-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. The following applies to BLOBs only: If IXFCSBCP is zero, the string is bit data, and should not be translated by any transformation program. The following applies to CLOBs only: If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP.

Table 21. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
BLOB_LOCATION_ SPECIFIER and DBCLOB_ LOCATION_ SPECIFIER	960, 964, 968	A fixed-length field, which cannot exceed 255 bytes. The LOB Location Specifier (LLS)is located in the code page indicated by IXFCSBCP. If IXFCSBCP is zero, the LLS is bit data and should not be translated by any transformation program. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. Since the length of the LLS is stored in IXFCLENG, the actual length of the original LOB is lost. PC/IXF files with columns of this type should not be used to recreate the LOB field since the LOB will be created with the length of the LLS.
BLOB_FILE, CLOB_FILE, DBCLOB_FILE	916, 920, 924	A fixed-length field containing an SQLFILE structure with the *name_length* and the *name* fields filled in. The length of the structure is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 255 bytes. The file name is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the file name can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the file name is bit data and should not be translated by any transformation program. Since the length of the structure is stored in IXFCLENG, the actual length of the original LOB is lost. IXF files with columns of type BLOB_FILE, CLOB_FILE, or DBCLOB_FILE should not be used to recreate the LOB field, since the LOB will be created with a length of *sql_lobfile_len*.
CHAR	452	A fixed-length character string. The string length is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 254 bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.

Table 21. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
DATE	384	A point in time in accordance with the Gregorian calendar. Each date is a 10-byte character string in International Standards Organization (ISO) format: *yyyy-mm-dd*. The range of the year part is `0001` to `9999`. The range of the month part is `01` to `12`. The range of the day part is `01` to *n*, where *n* depends on the month, using the usual rules for days of the month and leap year. Leading zeros cannot be omitted from any part. IXFCLENG is not used, and should contain blanks. Valid characters within DATE are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
DBCLOB	412	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters in the string, and cannot exceed 16 383. The string itself is preceded by a current length indicator, which is a 4-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
DECIMAL	484	A packed decimal number with precision P (as specified by the first three bytes of IXFCLENG in the column descriptor record) and scale S (as specified by the last two bytes of IXFCLENG). The length, in bytes, of a packed decimal number is (P+2)/2. The precision must be an odd number between 1 and 31, inclusive. The packed decimal number is in the internal format specified by IXFTMFRM, where packed decimal for the PC is defined to be the same as packed decimal for the System/370. IXFCSBCP and IXFCDBCP are not significant, and should be zero.
FLOATING POINT	480	Either a long (8-byte) or short (4-byte) floating point number, depending on whether IXFCLENG is set to eight or to four. The data is in the internal machine form, as specified by IXFTMFRM. IXFCSBCP and IXFCDBCP are not significant, and should be zero. Four-byte floating point is not supported by the database manager.

Table 21. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
GRAPHIC	468	A fixed-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the number of double-byte characters in the string, and cannot exceed 127. The actual length of the string is twice the value of the IXFCLENG field, in bytes. The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
INTEGER	496	A 4-byte integer in the form specified by IXFTMFRM. It represents a whole number between -2 147 483 648 and +2 147 483 647. IXFCSBCP and IXFCDBCP are not significant, and should be zero. IXFCLENG is not used, and should contain blanks.
LONGVARCHAR	456	A variable-length character string. The maximum length of the string is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 32 767 bytes. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.
LONG VARGRAPHIC	472	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters for the string, and cannot exceed 16 383. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.
SMALLINT	500	A 2-byte integer in the form specified by IXFTMFRM. It represents a whole number between –32 768 and +32 767. IXFCSBCP and IXFCDBCP are not significant, and should be zero. IXFCLENG is not used, and should contain blanks.

Table 21. PC/IXF Data Types (continued)

Name	IXFCTYPE Value	Description
TIME	388	A point in time in accordance with the 24-hour clock. Each time is an 8-byte character string in ISO format: *hh.mm.ss*. The range of the hour part is `00` to `24`, and the range of the other parts is `00` to `59`. If the hour is `24`, the other parts are `00`. The smallest time is `00.00.00`, and the largest is `24.00.00`. Leading zeros cannot be omitted from any part. IXFCLENG is not used, and should contain blanks. Valid characters within TIME are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
TIMESTAMP	392	The date and time with microsecond precision. Each time stamp is a character string of the form *yyyy-mm-dd-hh.mm.ss.nnnnnn* (year month day hour minutes seconds microseconds). IXFCLENG is not used, and should contain blanks. Valid characters within TIMESTAMP are invariant in all PC ASCII code pages; therefore, IXFCSBCP and IXFCDBCP are not significant, and should be zero.
VARCHAR	448	A variable-length character string. The maximum length of the string, in bytes, is contained in the IXFCLENG field of the column descriptor record, and cannot exceed 254 bytes. The string itself is preceded by a current length indicator, which is a two-byte integer specifying the length of the string, in bytes. The string is in the code page indicated by IXFCSBCP. If IXFCDBCP is non-zero, the string can also contain double-byte characters in the code page indicated by IXFCDBCP. If IXFCSBCP is zero, the string is bit data and should not be translated by any transformation program.
VARGRAPHIC	464	A variable-length string of double-byte characters. The IXFCLENG field in the column descriptor record specifies the maximum number of double-byte characters in the string, and cannot exceed 127. The string itself is preceded by a current length indicator, which is a 2-byte integer specifying the length of the string in double-byte characters (that is, the value of this integer is one half the length of the string, in bytes). The string is in the DBCS code page, as specified by IXFCDBCP in the C record. Since the string consists of double-byte character data only, IXFCSBCP should be zero. There are no surrounding shift-in or shift-out characters.

Not all combinations of IXFCSBCP and IXFCDBCP values for PC/IXF character or graphic columns are valid. A PC/IXF character or graphic column with an invalid (IXFCSBCP,IXFCDBCP) combination is an invalid data type.

Table 22. Valid PC/IXF Data Types

PC/IXF Data Type	Valid (IXFCSBCP,IXFCDBCP) Pairs	Invalid (IXFCSBCP,IXFCDBCP) Pairs
CHAR, VARCHAR, or LONG VARCHAR	(0,0), (x,0), or (x,y)	(0,y)
BLOB	(0,0)	(x,0), (0,y), or (x,y)
CLOB	(x,0), (x,y)	(0,0), (0,y)
GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, or DBCLOB	(0,y)	(0,0), (x,0), or (x,y)
Note: x and y are not 0.		

Related reference:

- “FORCEIN Option” on page 332
- “PC/IXF Data Type Descriptions” on page 325
- “PC/IXF Record Types” on page 304

PC/IXF Data Type Descriptions

Table 23. Acceptable Data Type Forms for the PC/IXF File Format

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
BIGINT	A BIGINT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -9 223 372 036 854 775 808 to 9 223 372 036 854 775 807.
BLOB	A PC/IXF BLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, LONG VARCHAR, BLOB, BLOB_FILE, or BLOB_LOCATION_SPECIFIER column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC BLOB column is also acceptable. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

Table 23. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	Form in Files Created by the Export Utility	Form Acceptable to the Import Utility
CHAR	A PC/IXF CHAR column is created. The database column length, the SBCS CPGID value, and the DBCS CPGID value are copied to the PC/IXF column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the length of the database column. The data is padded on the right with single-byte spaces (x'20'), if necessary.
CLOB	A PC/IXF CLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, LONG VARCHAR, CLOB, CLOB_FILE, or CLOB_LOCATION_SPECIFIER column is acceptable if the PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
DATE	A DATE column, identical to the database column, is created.	A PC/IXF column of type DATE is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain dates in a format consistent with the territory code of the target database.
DBCLOB	A PC/IXF DBCLOB column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, LONG VARGRAPHIC, DBCLOB, DBCLOB_FILE, or DBCLOB_LOCATION_SPECIFIER column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
DECIMAL	A DECIMAL column, identical to the database column, is created. The precision and scale of the column is stored in the column descriptor record.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range of the DECIMAL column into which they are being imported.
FLOAT	A FLOAT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. All values are within range.

Table 23. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	**Form in Files Created by the Export Utility**	**Form Acceptable to the Import Utility**
GRAPHIC (DBCS only)	A PC/IXF GRAPHIC column is created. The database column length, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the database column length. The data is padded on the right with double-byte spaces (x'8140'), if necessary.
INTEGER	An INTEGER column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -2 147 483 648 to 2 147 483 647.
LONG VARCHAR	A PC/IXF LONG VARCHAR column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
LONG VARGRAPHIC (DBCS only)	A PC/IXF LONG VARGRAPHIC column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
SMALLINT	A SMALLINT column, identical to the database column, is created.	A column in any numeric type (SMALLINT, INTEGER, BIGINT, DECIMAL, or FLOAT) is accepted. Individual values are rejected if they are not in the range -32 768 to 32 767.
TIME	A TIME column, identical to the database column, is created.	A PC/IXF column of type TIME is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain time data in a format consistent with the territory code of the target database.
TIMESTAMP	A TIMESTAMP column, identical to the database column, is created.	A PC/IXF column of type TIMESTAMP is the usual input. The import utility also attempts to accept columns in any of the character types, except those with incompatible lengths. The character column in the PC/IXF file must contain data in the input format for time stamps.

Table 23. Acceptable Data Type Forms for the PC/IXF File Format (continued)

Data Type	**Form in Files Created by the Export Utility**	**Form Acceptable to the Import Utility**
VARCHAR	If the maximum length of the database column is <= 254, a PC/IXF VARCHAR column is created. If the maximum length of the database column is > 254, a PC/IXF LONG VARCHAR column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF CHAR, VARCHAR, or LONG VARCHAR column is acceptable if: • The database column is marked FOR BIT DATA • The PC/IXF column single-byte code page value equals the SBCS CPGID of the database column, and the PC/IXF column double-byte code page value equals zero, or the DBCS CPGID of the database column. A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is also acceptable if the database column is marked FOR BIT DATA. In any case, if the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.
VARGRAPHIC (DBCS only)	If the maximum length of the database column is <= 127, a PC/IXF VARGRAPHIC column is created. If the maximum length of the database column is > 127, a PC/IXF LONG VARGRAPHIC column is created. The maximum length of the database column, the SBCS CPGID value, and the DBCS CPGID value are copied to the column descriptor record.	A PC/IXF GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC column is acceptable if the PC/IXF column double-byte code page value equals that of the database column. If the PC/IXF column is of fixed length, its length must be compatible with the maximum length of the database column.

Related reference:

- "PC/IXF data types" on page 320
- "PC/IXF Record Types" on page 304

General Rules Governing PC/IXF File Import into Databases

The database manager import utility applies the following general rules when importing a PC/IXF file in either an SBCS or a DBCS environment:

- The import utility accepts PC/IXF format files only (IXFHID = 'IXF'). IXF files of other formats cannot be imported.
- The import utility rejects a PC/IXF file with more than 1024 columns.
- The value of IXFHSBCP in the PC/IXF H record must equal the SBCS CPGID, or there must be a conversion table between the IXFHSBCP/IXFHDBCP and the SBCS/DBCS CPGID of the target database. The value of IXFHDBCP must equal either '00000', or the DBCS CPGID of the target database. If either of these conditions is not satisfied, the import utility rejects the PC/IXF file, unless the FORCEIN option is specified.
- Invalid Data Types — New Table

  Import of a PC/IXF file into a *new* table is specified by the CREATE or the REPLACE_CREATE keywords in the IMPORT command. If a PC/IXF column of an invalid data type is selected for import into a new table, the import utility terminates. The entire PC/IXF file is rejected, no table is created, and no data is imported.

- Invalid Data Types — Existing Table

  Import of a PC/IXF file into an *existing* table is specified by the INSERT, the INSERT_UPDATE, the REPLACE or the REPLACE_CREATE keywords in the IMPORT command. If a PC/IXF column of an invalid data type is selected for import into an existing table, one of two actions is possible:
  - If the target table column is nullable, all values for the invalid PC/IXF column are ignored, and the table column values are set to NULL
  - If the target table column is not nullable, the import utility terminates. The entire PC/IXF file is rejected, and no data is imported. The existing table remains unaltered.
- When importing into a new table, nullable PC/IXF columns generate nullable database columns, and not nullable PC/IXF columns generate not nullable database columns.
- A not nullable PC/IXF column can be imported into a nullable database column.
- A nullable PC/IXF column can be imported into a not nullable database column. If a NULL value is encountered in the PC/IXF column, the import utility rejects the values of all columns in the PC/IXF row that contains the NULL value (the entire row is rejected), and processing continues with the next PC/IXF row. That is, no data is imported from a PC/IXF row that contains a NULL value if a target table column (for the NULL) is not nullable.
- Incompatible Columns — New Table

  If, during import to a *new* database table, a PC/IXF column is selected that is incompatible with the target database column, the import utility terminates. The entire PC/IXF file is rejected, no table is created, and no data is imported.

  Note: The IMPORT FORCEIN option extends the scope of compatible columns.
- Incompatible Columns — Existing Table

  If, during import to an *existing* database table, a PC/IXF column is selected that is incompatible with the target database column, one of two actions is possible:
  - If the target table column is nullable, all values for the PC/IXF column are ignored, and the table column values are set to NULL
  - If the target table column is not nullable, the import utility terminates. The entire PC/IXF file is rejected, and no data is imported. The existing table remains unaltered.

  Note: The IMPORT FORCEIN option extends the scope of compatible columns.
- Invalid Values

  If, during import, a PC/IXF column value is encountered that is not valid for the target database column, the import utility rejects the values of all columns in the PC/IXF row that contains the invalid value (the entire row is rejected), and processing continues with the next PC/IXF row.

Related reference:
- "PC/IXF data types" on page 320
- "FORCEIN Option" on page 332

Data Type-Specific Rules Governing PC/IXF File Import into Databases

- A valid PC/IXF numeric column can be imported into any compatible numeric database column. PC/IXF columns containing 4-byte floating point data are not imported, because this is an invalid data type.
- Database date/time columns can accept values from matching PC/IXF date/time columns (DATE, TIME, and TIMESTAMP), as well as from PC/IXF character columns (CHAR, VARCHAR, and LONG VARCHAR), subject to column length and value compatibility restrictions.
- A valid PC/IXF character column (CHAR, VARCHAR, or LONG VARCHAR) can always be imported into an *existing* database character column marked FOR BIT DATA; otherwise:
  - IXFCSBCP and the SBCS CPGID must agree
  - There must be a conversion table for the IXFCSBCP/IXFCDBCP and the SBCS/DBCS
  - One set must be all zeros (FOR BIT DATA).

  If IXFCSBCP is not zero, the value of IXFCDBCP must equal either zero or the DBCS CPGID of the target database column.

  If either of these conditions is not satisfied, the PC/IXF and database columns are incompatible.

  When importing a valid PC/IXF character column into a *new* database table, the value of IXFCSBCP must equal either zero or the SBCS CPGID of the database, or there must be a conversion table. If IXFCSBCP is zero, IXFCDBCP must also be zero (otherwise the PC/IXF column is an invalid data type); IMPORT creates a character column marked FOR BIT DATA in the new table. If IXFCSBCP is not zero, and equals the SBCS CPGID of the database, the value of IXFCDBCP must equal either zero or the DBCS CPGID of the database; in this case, the utility creates a character column in the new table with SBCS and DBCS CPGID values equal to those of the database. If these conditions are not satisfied, the PC/IXF and database columns are incompatible.

  The FORCEIN option can be used to override code page equality checks. However, a PC/IXF character column with IXFCSBCP equal to zero and IXFCDBCP not equal to zero is an invalid data type, and cannot be imported, even if FORCEIN is specified.
- A valid PC/IXF graphic column (GRAPHIC, VARGRAPHIC, or LONG VARGRAPHIC) can always be imported into an *existing* database character column marked FOR BIT DATA, but is incompatible with all other database columns. The FORCEIN option can be used to relax this restriction. However, a PC/IXF graphic column with IXFCSBCP not equal to zero, or IXFCDBCP equal to zero, is an invalid data type, and cannot be imported, even if FORCEIN is specified.

  When importing a valid PC/IXF graphic column into a database graphic column, the value of IXFCDBCP must equal the DBCS CPGID of the target database column (that is, the double-byte code pages of the two columns must agree).
- If, during import of a PC/IXF file into an existing database table, a fixed-length string column (CHAR or GRAPHIC) is selected whose length is greater than the maximum length of the target column, the columns are incompatible.
- If, during import of a PC/IXF file into an existing database table, a variable-length string column (VARCHAR, LONG VARCHAR, VARGRAPHIC, or LONG VARGRAPHIC) is selected whose length is greater than the maximum length of the target column, the columns *are* compatible. Individual values are

processed according to the compatibility rules governing the database manager INSERT statement, and PC/IXF values which are too long for the target database column are invalid.

- PC/IXF values imported into a fixed-length database *character* column (that is, a CHAR column) are padded on the right with single-byte spaces (0x20), if necessary, to obtain values whose length equals that of the database column. PC/IXF values imported into a fixed-length database *graphic* column (that is, a GRAPHIC column) are padded on the right with double-byte spaces (0x8140), if necessary, to obtain values whose length equals that of the database column.
- Since PC/IXF VARCHAR columns have a maximum length of 254 bytes, a database VARCHAR column of maximum length *n*, with 254 < *n* < 4001, must be exported into a PC/IXF LONG VARCHAR column of maximum length *n*.
- Although PC/IXF LONG VARCHAR columns have a maximum length of 32 767 bytes, and database LONG VARCHAR columns have a maximum length restriction of 32 700 bytes, PC/IXF LONG VARCHAR columns of length greater than 32 700 bytes (but less than 32 768 bytes) are still valid, and can be imported into database LONG VARCHAR columns, but data might be lost.
- Since PC/IXF VARGRAPHIC columns have a maximum length of 127 bytes, a database VARGRAPHIC column of maximum length *n*, with 127 < *n* < 2001, must be exported into a PC/IXF LONG VARGRAPHIC column of maximum length *n*.
- Although PC/IXF LONG VARGRAPHIC columns have a maximum length of 16 383 bytes, and database LONG VARGRAPHIC columns have a maximum length restriction of 16 350, PC/IXF LONG VARGRAPHIC columns of length greater than 16 350 bytes (but less than 16 384 bytes) are still valid, and can be imported into database LONG VARGRAPHIC columns, but data might be lost.

Table 24 summarizes PC/IXF file import into new or existing database tables without the FORCEIN option.

Table 24. Summary of PC/IXF File Import without FORCEIN Option

	DATABASE COLUMN DATA TYPE											
PC/IXF COLUMN DATA TYPE	**SMALL INT**	**INT**	**BIGINT**	**DEC**	**FLT**	**(0,0)**	**(SBCS, 0)[d]**	**(SBCS, DBCS)[b]**	**GRAPH[b]**	**DATE**	**TIME**	**TIME STAMP**
-SMALLINT	N											
	E	E	E	E[a]	E							
-INTEGER		N										
	E[a]	E	E	E[a]	E							
-BIGINT			N									
	E[a]	E[a]	E	E[a]	E							
-DECIMAL				N								
	E[a]	E[a]	E[a]	E[a]	E							
-FLOAT					N							
	E[a]	E[a]	E[a]	E[a]	E							
-(0,0)						N						
						E				E[c]	E[c]	E[c]
-(SBCS,0)							N	N				
						E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)								N		E[c]	E[c]	E[c]
						E		E				
-GRAPHIC									N			

Table 24. Summary of PC/IXF File Import without FORCEIN Option (continued)

	DATABASE COLUMN DATA TYPE											
PC/IXF COLUMN DATA TYPE	SMALL INT	INT	BIGINT	DEC	FLT	(0,0)	(SBCS, 0)[d]	(SBCS, DBCS)[b]	GRAPH[b]	DATE	TIME	TIME STAMP
						E			E			
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

Notes:

1. The table is a matrix of all valid PC/IXF and database manager data types. If a PC/IXF column can be imported into a database column, a letter is displayed in the matrix cell at the intersection of the PC/IXF data type matrix row and the database manager data type matrix column. An 'N' indicates that the utility is creating a new database table (a database column of the indicated data type is created). An 'E' indicates that the utility is importing data to an existing database table (a database column of the indicated data type is a valid target).
2. Character string data types are distinguished by code page attributes. These attributes are shown as an ordered pair (SBCS,DBCS), where:
   - SBCS is either zero or denotes a non-zero value of the single-byte code page attribute of the character data type
   - DBCS is either zero or denotes a non-zero value of the double-byte code page attribute of the character data type.
3. If the table indicates that a PC/IXF character column can be imported into a database character column, the values of their respective code page attribute pairs satisfy the rules governing code page equality.

[a] Individual values are rejected if they are out of range for the target numeric data type.

[b] Data type is available only in DBCS environments.

[c] Individual values are rejected if they are not valid date or time values.

[d] Data type is not available in DBCS environments.

Related reference:

- "PC/IXF data types" on page 320
- "PC/IXF Data Type Descriptions" on page 325
- "General Rules Governing PC/IXF File Import into Databases" on page 328
- "PC Version of IXF File Format" on page 302
- "PC/IXF Record Types" on page 304

FORCEIN Option

The FORCEIN option permits import of a PC/IXF file despite code page differences between data in the PC/IXF file and the target database. It offers additional flexibility in the definition of compatible columns.

FORCEIN General Semantics

The following general semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment:

- The FORCEIN option should be used with caution. It is usually advisable to attempt an import without this option enabled. However, because of the generic nature of the PC/IXF data interchange architecture, some PC/IXF files might contain data types or values that cannot be imported without intervention.
- Import with FORCEIN to a *new* table might yield a different result than import to an existing table. An existing table has predefined target data types for each PC/IXF data type.

- When LOB data is exported with the LOBSINFILE option, and the files move to another client with a different code page, then, unlike other data, the CLOBS and DBCLOBS in the separate files are not converted to the client code page when imported or loaded into a database.

FORCEIN Code Page Semantics

The following code page semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment:

- The FORCEIN option disables all import utility code page comparisons.

  This rule applies to code page comparisons at the column level and at the file level as well, when importing to a new or an existing database table. At the column (for example, data type) level, this rule applies only to the following database manager and PC/IXF data types: character (CHAR, VARCHAR, and LONG VARCHAR), and graphic (GRAPHIC, VARGRAPHIC, and LONG VARGRAPHIC). The restriction follows from the fact that code page attributes of other data types are not relevant to the interpretation of data type values.
- The FORCEIN option does not disable inspection of code page attributes to determine data types.

  For example, the database manager allows a CHAR column to be declared with the FOR BIT DATA attribute. Such a declaration sets both the SBCS CPGID and the DBCS CPGID of the column to zero; it is the zero value of these CPGIDs that identifies the column values as bit strings (rather than character strings).
- The FORCEIN option does not imply code page translation.

  Values of data types that are sensitive to the FORCEIN option are copied "as is". No code point mappings are employed to account for a change of code page environments. Padding of the imported value with spaces might be necessary in the case of fixed length target columns.
- When data is imported to an *existing* table using the FORCEIN option:
  - The code page value of the target database table and columns always prevails.
  - The code page value of the PC/IXF file and columns is ignored.

  This rule applies whether or not the FORCEIN option is used. The database manager does not permit changes to a database or a column code page value once a database is created.
- When importing to a *new* table using the FORCEIN option:
  - The code page value of the target database prevails.
  - PC/IXF character columns with IXFCSBCP = IXFCDBCP = 0 generate table columns marked FOR BIT DATA.
  - All other PC/IXF character columns generate table character columns with SBCS and DBCS CPGID values equal to those of the database.
  - PC/IXF graphic columns generate table graphic columns with an SBCS CPGID of "undefined", and a DBCS CPGID equal to that of the database (DBCS environment only).

FORCEIN Example

Consider a PC/IXF CHAR column with IXFCSBCP = ’00897’ and IXFCDBCP = ’00301’. This column is to be imported into a database CHAR column whose SBCS CPGID = ’00850’ and DBCS CPGID = ’00000’. Without FORCEIN, the utility terminates, and no data is imported, or the PC/IXF column values are ignored, and the database column contains NULLs (if the database column is nullable). With FORCEIN, the utility proceeds, ignoring code page incompatibilities. If there are no other data type incompatibilities (such as length, for example), the values of the PC/IXF column are imported "as is", and become available for interpretation under the database column code page environment.

The following two tables show:

- The code page attributes of a column created in a *new* database table when a PC/IXF file data type with specified code page attributes is imported.
- That the import utility rejects PC/IXF data types if they are invalid or incompatible.

Table 25. Summary of Import Utility Code Page Semantics (New Table) for SBCS. This table assumes there is no conversion table between a and x. If there were, items 3 and 4 would work successfully without the FORCEIN option.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	**CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN**	
	Without FORCEIN	**With FORCEIN**
(0,0)	(0,0)	(0,0)
(a,0)	(a,0)	(a,0)
(x,0)	reject	(a,0)
(x,y)	reject	(a,0)
(a,y)	reject	(a,0)
(0,y)	reject	(0,0)

Notes:

1. See the notes for Table 26.

Table 26. Summary of Import Utility Code Page Semantics (New Table) for DBCS. This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	**CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN**	
	Without FORCEIN	**With FORCEIN**
(0,0)	(0,0)	(0,0)
(a,0)	(a,b)	(a,b)
(x,0)	reject	(a,b)
(a,b)	(a,b)	(a,b)
(x,y)	reject	(a,b)
(a,y)	reject	(a,b)
(x,b)	reject	(a,b)
(0,b)	(-,b)	(-,b)
(0,y)	reject	(-,b)

Table 26. Summary of Import Utility Code Page Semantics (New Table) for DBCS (continued). This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES of PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF DATABASE TABLE COLUMN	
	Without FORCEIN	With FORCEIN

Notes:

1. Code page attributes of a PC/IXF data type are shown as an ordered pair, where x represents a non-zero single-byte code page value, and y represents a non-zero double-byte code page value. A '-' represents an undefined code page value.
2. The use of different letters in various code page attribute pairs is deliberate. Different letters imply different values. For example, if a PC/IXF data type is shown as (x,y), and the database column as (a,y), x does not equal a, but the PC/IXF file and the database have the same double-byte code page value y.
3. Only character and graphic data types are affected by the FORCEIN code page semantics.
4. It is assumed that the database containing the new table has code page attributes of (a,0); therefore, all character columns in the new table must have code page attributes of either (0,0) or (a,0).

   In a DBCS environment, it is assumed that the database containing the new table has code page attributes of (a,b); therefore, all graphic columns in the new table must have code page attributes of (-,b), and all character columns must have code page attributes of (a,b). The SBCS CPGID is shown as '-', because it is undefined for graphic data types.
5. The data type of the result is determined by the rules described in "FORCEIN Data Type Semantics" on page 337.
6. The `reject` result is a reflection of the rules for invalid or incompatible data types.

The following two tables show:

- That the import utility accepts PC/IXF data types with various code page attributes into an *existing* table column (the *target* column) having the specified code page attributes.
- That the import utility does not permit a PC/IXF data type with certain code page attributes to be imported into an *existing* table column having the code page attributes shown. The utility rejects PC/IXF data types if they are invalid or incompatible.

Table 27. Summary of Import Utility Code Page Semantics (Existing Table) for SBCS. This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without FORCEIN	With FORCEIN
(0,0)	(0,0)	accept	accept
(a,0)	(0,0)	accept	accept
(x,0)	(0,0)	accept	accept
(x,y)	(0,0)	accept	accept
(a,y)	(0,0)	accept	accept
(0,y)	(0,0)	accept	accept

Table 27. Summary of Import Utility Code Page Semantics (Existing Table) for SBCS (continued). This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without FORCEIN	**With FORCEIN**
(0,0)	(a,0)	null or reject	accept
(a,0)	(a,0)	accept	accept
(x,0)	(a,0)	null or reject	accept
(x,y)	(a,0)	null or reject	accept
(a,y)	(a,0)	null or reject	accept
(0,y)	(a,0)	null or reject	null or reject

Notes:

1. See the notes for Table 25 on page 334.
2. The `null or reject` result is a reflection of the rules for invalid or incompatible data types.

Table 28. Summary of Import Utility Code Page Semantics (Existing Table) for DBCS. This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without FORCEIN	**With FORCEIN**
(0,0)	(0,0)	accept	accept
(a,0)	(0,0)	accept	accept
(x,0)	(0,0)	accept	accept
(a,b)	(0,0)	accept	accept
(x,y)	(0,0)	accept	accept
(a,y)	(0,0)	accept	accept
(x,b)	(0,0)	accept	accept
(0,b)	(0,0)	accept	accept
(0,y)	(0,0)	accept	accept
(0,0)	(a,b)	null or reject	accept
(a,0)	(a,b)	accept	accept
(x,0)	(a,b)	null or reject	accept
(a,b)	(a,b)	accept	accept
(x,y)	(a,b)	null or reject	accept
(a,y)	(a,b)	null or reject	accept
(x,b)	(a,b)	null or reject	accept
(0,b)	(a,b)	null or reject	null or reject
(0,y)	(a,b)	null or reject	null or reject

Table 28. Summary of Import Utility Code Page Semantics (Existing Table) for DBCS (continued). This table assumes there is no conversion table between a and x.

CODE PAGE ATTRIBUTES OF PC/IXF DATA TYPE	CODE PAGE ATTRIBUTES OF TARGET DATABASE COLUMN	RESULTS OF IMPORT	
		Without FORCEIN	**With FORCEIN**
(0,0)	(-,b)	null or reject	accept
(a,0)	(-,b)	null or reject	null or reject
(x,0)	(-,b)	null or reject	null or reject
(a,b)	(-,b)	null or reject	null or reject
(x,y)	(-,b)	null or reject	null or reject
(a,y)	(-,b)	null or reject	null or reject
(x,b)	(-,b)	null or reject	null or reject
(0,b)	(-,b)	accept	accept
(0,y)	(-,b)	null or reject	accept

Notes:

1. See the notes for Table 25 on page 334.
2. The `null or reject` result is a reflection of the rules for invalid or incompatible data types.

FORCEIN Data Type Semantics

The FORCEIN option permits import of certain PC/IXF columns into target database columns of unequal and otherwise incompatible data types. The following data type semantics apply when using the FORCEIN option in either an SBCS or a DBCS environment (except where noted):

- In SBCS environments, the FORCEIN option permits import of:
  - A PC/IXF BIT data type (IXFCSBCP = 0 = IXFCDBCP for a PC/IXF character column) into a database character column (non-zero SBCS CPGID, and DBCS CPGID = 0); existing tables only
  - A PC/IXF MIXED data type (non-zero IXFCSBCP and IXFCDBCP) into a database character column; both new and existing tables
  - A PC/IXF GRAPHIC data type into a database FOR BIT DATA column (SBCS CPGID = 0 = DBCS CPGID); new tables only (this is always permitted for existing tables).
- The FORCEIN option does not extend the scope of valid PC/IXF data types.

  PC/IXF columns with data types not defined as valid PC/IXF data types are invalid for import with or without the FORCEIN option.
- In DBCS environments, the FORCEIN option permits import of:
  - A PC/IXF BIT data type into a database character column
  - A PC/IXF BIT data type into a database graphic column; however, if the PC/IXF BIT column is of fixed length, that length must be even. A fixed length PC/IXF BIT column of odd length is not compatible with a database graphic column. A varying-length PC/IXF BIT column *is* compatible whether its length is odd or even, although an odd-length value from a varying-length column is an invalid value for import into a database graphic column
  - A PC/IXF MIXED data type into a database character column.

Table 29 summarizes PC/IXF file import into new or existing database tables with the FORCEIN option.

Table 29. Summary of PC/IXF File Import with FORCEIN Option

	DATABASE COLUMN DATA TYPE											
PC/IXF COLUMN DATA TYPE	**SMALL INT**	**INT**	**BIGINT**	**DEC**	**FLT**	**(0,0)**	**(SBCS, 0)[e]**	**(SBCS, DBCS)[b]**	**GRAPH[b]**	**DATE**	**TIME**	**TIME STAMP**
-SMALLINT	N											
	E	E	E	E[a]	E							
-INTEGER		N										
	E[a]	E	E	E[a]	E							
-BIGINT			N									
	E[a]	E[a]	E	E[a]	E							
-DECIMAL				N								
	E[a]	E[a]	E[a]	E[a]	E							
-FLOAT					N							
	E[a]	E[a]	E[a]	E[a]	E							
-(0,0)						N						
						E	E w/F	E w/F	E w/F	E[c]	E[c]	E[c]
-(SBCS,0)							N	N				
						E	E	E		E[c]	E[c]	E[c]
-(SBCS, DBCS)							N w/F[d]	N		E[c]	E[c]	E[c]
						E	E w/F	E				
-GRAPHIC						N w/F[d]			N			
						E			E			
-DATE										N		
										E		
-TIME											N	
											E	
-TIME STAMP												N
												E

Note: If a PC/IXF column can be imported into a database column only with the FORCEIN option, the string 'w/F' is displayed together with an 'N' or an 'E'. An 'N' indicates that the utility is creating a new database table; an 'E' indicates that the utility is importing data to an existing database table. The FORCEIN option affects compatibility of character and graphic data types only.

[a] Individual values are rejected if they are out of range for the target numeric data type.

[b] Data type is available only in DBCS environments.

[c] Individual values are rejected if they are not valid date or time values.

[d] Applies only if the source PC/IXF data type is not supported by the target database.

[e] Data type is not available in DBCS environments.

Related reference:

- "PC/IXF data types" on page 320
- "General Rules Governing PC/IXF File Import into Databases" on page 328

Differences Between PC/IXF and Version 0 System/370 IXF

The following describes differences between PC/IXF, used by the database manager, and Version 0 System/370 IXF, used by several host database products:

- PC/IXF files are ASCII, rather than EBCDIC oriented. PC/IXF files have significantly expanded code page identification, including new code page identifiers in the H record, and the use of actual code page values in the column descriptor records. There is also a mechanism for marking columns of character data as FOR BIT DATA. FOR BIT DATA columns are of special significance, because transforms which convert a PC/IXF file format to or from any other IXF or database file format cannot perform any code page translation on the values contained in FOR BIT DATA columns.
- Only the machine data form is permitted; that is, the IXFTFORM field must always contain the value `M`. Furthermore, the machine data must be in PC forms; that is, the IXFTMFRM field must contain the value `PC`. This means that integers, floating point numbers, and decimal numbers in data portions of PC/IXF data records must be in PC forms.
- Application (A) records are permitted anywhere after the H record in a PC/IXF file. They are not counted when the value of the IXFHHCNT field is computed.
- Every PC/IXF record begins with a record length indicator. This is a 6-byte character representation of an integer value containing the length, in bytes, of the PC/IXF record not including the record length indicator itself; that is, the total record length minus 6 bytes. The purpose of the record length field is to enable PC programs to identify record boundaries.
- To facilitate the compact storage of variable-length data, and to avoid complex processing when a field is split into multiple records, PC/IXF does not support Version 0 IXF X records, but does support D record identifiers. Whenever a variable-length field or a nullable field is the last field in a data D record, it is not necessary to write the entire maximum length of the field to the PC/IXF file.

Related reference:
- "Data Type-Specific Rules Governing PC/IXF File Import into Databases" on page 330
- "General Rules Governing PC/IXF File Import into Databases" on page 328
- "PC/IXF data types" on page 320
- "PC/IXF Data Type Descriptions" on page 325

Worksheet File Format (WSF)

Lotus 1-2-3 and Symphony products use the same basic format, with additional functions added at each new release. The database manager supports the subset of the worksheet records that are the same for all the Lotus products. That is, for the releases of Lotus 1-2-3 and Symphony products supported by the database manager, all file names with any three-character extension are accepted; for example: WKS, WK1, WRK, WR1, WJ2.

Each WSF file represents one worksheet. The database manager uses the following conventions to interpret worksheets and to provide consistency in worksheets generated by its export operations:
- Cells in the first row (ROW value 0) are reserved for descriptive information about the entire worksheet. All data within this row is optional. It is ignored during import.
- Cells in the second row (ROW value 1) are used for column labels.
- The remaining rows are data rows (records, or rows of data from the table).
- Cell values under any column heading are values for that particular column or field.

- A NULL value is indicated by the absence of a real cell content record (for example, no integer, number, label, or formula record) for a particular column within a row of cell content records.

  Note: A row of NULLs will be neither imported nor exported.

To create a file that is compliant with the WSF format during an export operation, some loss of data might occur.

WSF files use a Lotus code point mapping that is not necessarily the same as existing code pages supported by DB2 database. As a result, when importing or exporting a WSF file, data is converted from the Lotus code points to or from the code points used by the application code page. DB2 supports conversion between the Lotus code points and code points defined by code pages 437, 819, 850, 860, 863, and 865.

Note: For multi-byte character set users, no conversions are performed.

Related concepts:
- "Moving data across platforms - file format considerations" on page 241

Appendix E. Export/Import/Load utility unicode considerations

The export, import, and load utilities are not supported when they are used with a Unicode client connected to a non-Unicode database. Unicode client files are only supported when the Unicode client is connected to a Unicode database.

The DEL, ASC, and PC/IXF file formats are supported for a UCS-2 database, as described in this section. The WSF format is not supported.

When exporting from a UCS-2 database to an ASCII delimited (DEL) file, all character data is converted to the application code page. Both character string and graphic string data are converted to the same SBCS or MBCS code page of the client. This is expected behavior for the export of any database, and cannot be changed, because the entire delimited ASCII file can have only one code page. Therefore, if you export to a delimited ASCII file, only those UCS-2 characters that exist in your application code page will be saved. Other characters are replaced with the default substitution character for the application code page. For UTF-8 clients (code page 1208), there is no data loss, because all UCS-2 characters are supported by UTF-8 clients.

When importing from an ASCII file (DEL or ASC) to a UCS-2 database, character string data is converted from the application code page to UTF-8, and graphic string data is converted from the application code page to UCS-2. There is no data loss. If you want to import ASCII data that has been saved under a different code page, you should change the data file code page before issuing the IMPORT command. One way to accomplish this is to set DB2CODEPAGE to the code page of the ASCII data file.

The range of valid ASCII delimiters for SBCS and MBCS clients is identical to what is currently supported by IBM DB2 V9.1 for those clients. The range of valid delimiters for UTF-8 clients is X’01’ to X’7F’, with the usual restrictions.

When exporting from a UCS-2 database to a PC/IXF file, character string data is converted to the SBCS/MBCS code page of the client. Graphic string data is not converted, and is stored in UCS-2 (code page 1200). There is no data loss.

When importing from a PC/IXF file to a UCS-2 database, character string data is assumed to be in the SBCS/MBCS code page stored in the PC/IXF header, and graphic string data is assumed to be in the DBCS code page stored in the PC/IXF header. Character string data is converted by the import utility from the code page specified in the PC/IXF header to the code page of the client, and then from the client code page to UTF-8 (by the INSERT statement). graphic string data is converted by the import utility from the DBCS code page specified in the PC/IXF header directly to UCS-2 (code page 1200).

The load utility places the data directly into the database and, by default, assumes data in ASC or DEL files to be in the code page of the database. Therefore, by default, no code page conversion takes place for ASCII files. When the code page for the data file has been explicitly specified (using the `codepage` modifier), the load utility uses this information to convert from the specified code page to the database code page before loading the data. For PC/IXF files, the load utility always converts from the code pages specified in the IXF header to the database code page (1208 for CHAR, and 1200 for GRAPHIC).

The code page for DBCLOB files is always 1200 for UCS-2. The code page for CLOB files is the same as the code page for the data files being imported, loaded or exported. For example, when loading or importing data using the PC/IXF format, the CLOB file is assumed to be in the code page specified by the PC/IXF header. If the DBCLOB file is in ASC or DEL format, the load utility assumes that CLOB data is in the code page of the database (unless explicitly specified otherwise using the `codepage` modifier), while the import utility assumes it to be in the code page of the client application.

The `nochecklengths` modifier is always specified for a UCS-2 database, because:
- Any SBCS can be connected to a database for which there is no DBCS code page
- Character strings in UTF-8 format usually have different lengths than those in client code pages.

Restrictions for Code Pages 1394, 1392 and 5488

The import, export and load utilities can now be used to transfer data from the new Chinese code page GB 18030 (code page identifier 1392 and 5488) and the new Japanese code page ShiftJISX 0213 (code page identifier 1394) to DB2 Unicode databases. In addition, the export utility can be used to transfer data from DB2 Unicode databases to GB 18030 or ShiftJIS X0213 code page data.

For example, the following command will load the Shift_JISX0213 data file `u/jp/user/x0213/data.del` residing on a remotely connected client into MYTABLE:

```
db2 load client from /u/jp/user/x0213/data.del
of del modified by codepage=1394 insert into mytable
```

where MYTABLE is located on a DB2 Unicode database.

Since only connections between a Unicode client and a Unicode server are supported, so you need to use either a Unicode client or set the DB2 registry variable DB2CODEPAGE to 1208 prior to using the load, import, or export utilities.

Conversion from code page 1394, 1392, or 5488 to Unicode can result in expansion. For example, a 2-byte character can be stored as two 16-bit Unicode characters in the GRAPHIC columns. You need to ensure the target columns in the Unicode database are wide enough to contain any expanded Unicode byte.

Restrictions for XML data movement

Native XML functionality is available for Unicode databases only. Use the USING CODESET option of the CREATE DATABASE command to specify a UTF-8 encoding for a new database.

Loading data into tables containing XML columns using the load utility is not supported. Data movement of XML data should be performed using the import and export utilities.

Incompatibilities

For applications connected to a UCS-2 database, graphic string data is always in UCS-2 (code page 1200). For applications connected to non-UCS-2 databases, the graphic string data is in the DBCS code page of the application, or not allowed if the application code page is SBCS. For example, when a 932 client is connected to a Japanese non-UCS-2 database, the graphic string data is in code page 301. For the 932 client applications connected to a UCS-2 database, the graphic string data is in UCS-2.

Related reference:

- "CREATE DATABASE command" in *Command Reference*
- "DEL Data Type Descriptions" on page 296
- "Non-delimited ASCII (ASC) file format" on page 299
- "PC Version of IXF File Format" on page 302
- "Restrictions on native XML data store" in *XML Guide*

Appendix F. Bind files used by the export, import and load utilities

The following table lists bind files with their default isolation levels, as well as which utilities use them and for what purpose.

Bind File (Default Isolation Level)	Utility/Purpose
db2ueiwi.bnd (CS)	Import/Export. Used to query information about table columns and indexes.
db2uexpm.bnd (CS)	Export. Used to fetch from the query specified for the export operation.
db2uimpm.bnd (RS)	Import. Used to insert data from the source data file into the target table when INSERT, REPLACE or REPLACE_CREATE option is used.
db2uipkg.bnd (CS)	Import. Used to check bind options.
db2uiici.bnd (RR)	Import. Used to create indexes when the IXF CREATE option is specified.
db2ucktb.bnd (CS)	Load. Used to perform general initialization processes for a load operation.
db2ulxld.bnd (CS)	Load. Used to process the query provided during a load from cursor operation.
db2uigsi.bnd (RS on UNIX based systems, RR on all other platforms)	Import/Export. Used to drop indexes and check for referential constraints for an import replace operation. Also used to retrieve identity column information for exporting IXF files.
db2uiict.bnd (RR)	Import. Used to create tables when the IXF CREATE option is specified.
db2uqtpd.bnd (RR)	Import/Export. Used to perform processing for hierarchical tables.
db2uqtnm.bnd (RR)	Import. Used to perform processing for hierarchical tables when the IXF CREATE option is specified.
db2uimtb.bnd (RS)	Import. Used to perform general initialization processes for an import operation.
db2ImpInsUpdate.bnd (RS)	Import. Used to insert data from the source data file into the target table when INSERT_UPDATE option is used. Cannot be bound with the INSERT BUF option.

Related concepts:

- "About isolation levels" in *Administration Guide: Planning*
- "Binding" in *Administration Guide: Planning*

Related tasks:

- "Binding utilities to the database" in *Administration Guide: Implementation*

Appendix G. Warning, error and completion messages

Messages generated by the various utilities are included among the SQL messages. These messages are generated by the database manager when a warning or error condition has been detected. Each message has a message identifier that consists of a prefix (SQL) and a four- or five-digit message number. There are three message types: notification, warning, and critical. Message identifiers ending with an `N` are error messages. Those ending with a `W` indicate warning or informational messages. Message identifiers ending with a `C` indicate critical system errors.

The message number is also referred to as the *SQLCODE*. The SQLCODE is passed to the application as a positive or negative number, depending on its message type (N, W, or C). N and C yield negative values, whereas W yields a positive value. DB2 returns the SQLCODE to the application, and the application can get the message associated with the SQLCODE. DB2 also returns an *SQLSTATE* value for conditions that could be the result of an SQL or XQuery statement. Some SQLCODE values have associated SQLSTATE values.

You can use the information contained in this topic to identify an error or problem, and to resolve the problem by using the appropriate recovery action. This information can also be used to understand where messages are generated and logged.

SQL messages, and the message text associated with SQLSTATE values, are also accessible from the operating system command line. To access help for these error messages, enter the following at the operating system command prompt:

```
db2 ? SQLnnnnn
```

where *nnnnn* represents the message number. On UNIX based systems, the use of double quotation mark delimiters is recommended; this will avoid problems if there are single character file names in the directory:

```
db2 "? SQLnnnnn"
```

The message identifier accepted as a parameter for the **db2** command is not case sensitive, and the terminating letter is not required. Therefore, the following commands will produce the same result:

```
db2 ? SQL0000N
db2 ? sql0000
db2 ? SQL0000n
```

If the message text is too long for your screen, use the following command (on UNIX based operating systems and others that support the ″more″ pipe):

```
db2 ? SQLnnnnn | more
```

You can also redirect the output to a file which can then be browsed.

Help can also be invoked from interactive input mode. To access this mode, enter the following at the operating system command prompt:

```
db2
```

To get DB2 message help in this mode, type the following at the command prompt (`db2 =>`):

```
? SQLnnnnn
```

The message text associated with SQLSTATEs can be retrieved by issuing:

```
db2 ? nnnnn
 or
db2 ? nn
```

where *nnnnn* is a five-character SQLSTATE value (alphanumeric), and *nn* is a two-digit SQLSTATE class code (the first two digits of the SQLSTATE value).

Related concepts:

- "Introduction to Messages" in *Message Reference Volume 1*

Appendix H. DB2 Database technical information

Overview of the DB2 technical information

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
  - Topics
  - Help for DB2 tools
  - Sample programs
  - Tutorials
- DB2 books
  - PDF files (downloadable)
  - PDF files (from the DB2 PDF CD)
  - printed books
- Command line help
  - Command help
  - Message help
- Sample programs

IBM periodically makes documentation updates available. If you access the online version on the DB2 Information Center at ibm.com®, you do not need to install documentation updates because this version is kept up-to-date by IBM. If you have installed the DB2 Information Center, it is recommended that you install the documentation updates. Documentation updates allow you to update the information that you installed from the *DB2 Information Center CD* or downloaded from Passport Advantage as new information becomes available.

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hard-copy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and Redbooks™ online at ibm.com. Access the DB2 Information Management software library site at http://www.ibm.com/software/data/sw-library/.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how we can improve the DB2 documentation, send an e-mail to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this e-mail address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

Related concepts:

- "Features of the DB2 Information Center" in *Online DB2 Information Center*
- "Sample files" in *Samples Topics*

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*
- "Updating the DB2 Information Center installed on your computer or intranet server" on page 355

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 350

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/shop/publications/order. DB2 Version 9 manuals in PDF format can be downloaded from www.ibm.com/software/data/db2/udb/support/manualsv9.html.

Although the tables identify books available in print, the books might not be available in your country or region.

The information in these books is fundamental to all DB2 users; you will find this information useful whether you are a programmer, a database administrator, or someone who works with DB2 Connect or other DB2 products.

Table 30. DB2 technical information

Name	Form Number	Available in print
Administration Guide: Implementation	SC10-4221	Yes
Administration Guide: Planning	SC10-4223	Yes
Administrative API Reference	SC10-4231	Yes
Administrative SQL Routines and Views	SC10-4293	No
Call Level Interface Guide and Reference, Volume 1	SC10-4224	Yes
Call Level Interface Guide and Reference, Volume 2	SC10-4225	Yes
Command Reference	SC10-4226	No
Data Movement Utilities Guide and Reference	SC10-4227	Yes
Data Recovery and High Availability Guide and Reference	SC10-4228	Yes
Developing ADO.NET and OLE DB Applications	SC10-4230	Yes
Developing Embedded SQL Applications	SC10-4232	Yes

Table 30. DB2 technical information (continued)

Name	Form Number	Available in print
Developing SQL and External Routines	SC10-4373	No
Developing Java Applications	SC10-4233	Yes
Developing Perl and PHP Applications	SC10-4234	No
Getting Started with Database Application Development	SC10-4252	Yes
Getting started with DB2 installation and administration on Linux and Windows	GC10-4247	Yes
Message Reference Volume 1	SC10-4238	No
Message Reference Volume 2	SC10-4239	No
Migration Guide	GC10-4237	Yes
Net Search Extender Administration and User's Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6842	Yes
Performance Guide	SC10-4222	Yes
Query Patroller Administration and User's Guide	GC10-4241	Yes
Quick Beginnings for DB2 Clients	GC10-4242	No
Quick Beginnings for DB2 Servers	GC10-4246	Yes
Spatial Extender and Geodetic Data Management Feature User's Guide and Reference	SC18-9749	Yes
SQL Guide	SC10-4248	Yes
SQL Reference, Volume 1	SC10-4249	Yes
SQL Reference, Volume 2	SC10-4250	Yes
System Monitor Guide and Reference	SC10-4251	Yes
Troubleshooting Guide	GC10-4240	No
Visual Explain Tutorial	SC10-4319	No
What's New	SC10-4253	Yes
XML Extender Administration and Programming	SC18-9750	Yes
XML Guide	SC10-4254	Yes
XQuery Reference	SC18-9796	Yes

Table 31. DB2 Connect-specific technical information

Name	Form Number	Available in print
DB2 Connect User's Guide	SC10-4229	Yes

Table 31. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print
Quick Beginnings for DB2 Connect Personal Edition	GC10-4244	Yes
Quick Beginnings for DB2 Connect Servers	GC10-4243	Yes

Table 32. WebSphere® Information Integration technical information

Name	Form Number	Available in print
WebSphere Information Integration: Administration Guide for Federated Systems	SC19-1020	Yes
WebSphere Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018	Yes
WebSphere Information Integration: Configuration Guide for Federated Data Sources	SC19-1034	No
WebSphere Information Integration: SQL Replication Guide and Reference	SC19-1030	Yes

Note: The DB2 Release Notes provide additional information specific to your product's release and fix pack level. For more information, see the related links.

Related concepts:

- "Overview of the DB2 technical information" on page 349
- "About the Release Notes" in *Release notes*

Related tasks:

- "Ordering printed DB2 books" on page 352

Ordering printed DB2 books

If you require printed DB2 books, you can buy them online in many but not all countries or regions. You can always order printed DB2 books from your local IBM representative. Keep in mind that some softcopy books on the *DB2 PDF Documentation* CD are unavailable in print. For example, neither volume of the *DB2 Message Reference* is available as a printed book.

Printed versions of many of the DB2 books available on the DB2 PDF Documentation CD can be ordered for a fee from IBM. Depending on where you are placing your order from, you may be able to order books online, from the IBM Publications Center. If online ordering is not available in your country or region, you can always order printed DB2 books from your local IBM representative. Note that not all books on the DB2 PDF Documentation CD are available in print.

Note: The most up-to-date and complete DB2 documentation is maintained in the DB2 Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Procedure:

To order printed DB2 books:

- To find out whether you can order printed DB2 books online in your country or region, check the IBM Publications Center at http://www.ibm.com/shop/publications/order. You must select a country, region, or language to access publication ordering information and then follow the ordering instructions for your location.
- To order printed DB2 books from your local IBM representative:
  - Locate the contact information for your local representative from one of the following Web sites:
    - The IBM directory of world wide contacts at www.ibm.com/planetwide
    - The IBM Publications Web site at http://www.ibm.com/shop/publications/order. You will need to select your country, region, or language to the access appropriate publications home page for your location. From this page, follow the ″About this site″ link.
  - When you call, specify that you want to order a DB2 publication.
  - Provide your representative with the titles and form numbers of the books that you want to order.

Related concepts:

- "Overview of the DB2 technical information" on page 349

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 350

Displaying SQL state help from the command line processor

DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure:

To invoke SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, `? 08003` displays help for the 08003 SQL state, and `? 08` displays help for the 08 class code.

Related tasks:

- "Invoking command help from the command line processor" in *Command Reference*
- "Invoking message help from the command line processor" in *Command Reference*

Accessing different versions of the DB2 Information Center

For DB2 Version 9 topics, the DB2 Information Center URL is http://publib.boulder.ibm.com/infocenter/db2luw/v9/.

For DB2 Version 8 topics, go to the Version 8 Information Center URL at: http://publib.boulder.ibm.com/infocenter/db2luw/v8/.

Related tasks:

- "Updating the DB2 Information Center installed on your computer or intranet server" on page 355

Displaying topics in your preferred language in the DB2 Information Center

The DB2 Information Center attempts to display topics in the language specified in your browser preferences. If a topic has not been translated into your preferred language, the DB2 Information Center displays the topic in English.

Procedure:

To display topics in your preferred language in the Internet Explorer browser:

1. In Internet Explorer, click the **Tools —> Internet Options —> Languages...** button. The Language Preferences window opens.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button.

     Note: Adding a language does not guarantee that the computer has the fonts required to display the topics in the preferred language.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

To display topics in your preferred language in a Firefox or Mozilla browser:

1. Select the **Tools —> Options —> Languages** button. The Languages panel is displayed in the Preferences window.
2. Ensure your preferred language is specified as the first entry in the list of languages.
   - To add a new language to the list, click the **Add...** button to select a language from the Add Languages window.
   - To move a language to the top of the list, select the language and click the **Move Up** button until the language is first in the list of languages.
3. Clear the browser cache and then refresh the page to display the DB2 Information Center in your preferred language.

On some browser and operating system combinations, you might have to also change the regional settings of your operating system to the locale and language of your choice.

Related concepts:

- "Overview of the DB2 technical information" on page 349

Updating the DB2 Information Center installed on your computer or intranet server

If you have a locally-installed DB2 Information Center, updated topics can be available for download. The 'Last updated' value found at the bottom of most topics indicates the current level for that topic.

To determine if there is an update available for the entire DB2 Information Center, look for the 'Last updated' value on the Information Center home page. Compare the value in your locally installed home page to the date of the most recent downloadable update at http://www.ibm.com/software/data/db2/udb/support/icupdate.html. You can then update your locally-installed Information Center if a more recent downloadable update is available.

Updating your locally-installed DB2 Information Center requires that you:

1. Stop the DB2 Information Center on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to download and apply updates.
2. Use the Update feature to determine if update packages are available from IBM.

   Note: Updates are also available on CD. For details on how to configure your Information Center to install updates from CD, see the related links.

   If update packages are available, use the Update feature to download the packages. (The Update feature is only available in stand-alone mode.)
3. Stop the stand-alone Information Center, and restart the DB2 Information Center service on your computer.

Procedure:

To update the DB2 Information Center installed on your computer or intranet server:

1. Stop the DB2 Information Center service.
   - On Windows, click **Start → Control Panel → Administrative Tools → Services**. Then right-click on **DB2 Information Center** service and select **Stop**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv9 stop
     ```
2. Start the Information Center in stand-alone mode.
   - On Windows:
     a. Open a command window.
     b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `C:\Program Files\IBM\DB2 Information Center\Version 9` directory.
     c. Run the `help_start.bat` file using the fully qualified path for the DB2 Information Center:

        ```
        <DB2 Information Center dir>\doc\bin\help_start.bat
        ```
   - On Linux:

a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V9` directory.
b. Run the `help_start` script using the fully qualified path for the DB2 Information Center:

```
<DB2 Information Center dir>/doc/bin/help_start
```

The systems default Web browser launches to display the stand-alone Information Center.

3. Click the Update button ( ). On the right hand panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the download process, check the selections you want to download, then click **Install Updates**.
5. After the download and installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center.
   - On Windows, run the `help_end.bat` file using the fully qualified path for the DB2 Information Center:

     ```
     <DB2 Information Center dir>\doc\bin\help_end.bat
     ```

     Note: The help_end batch file contains the commands required to safely terminate the processes that were started with the help_start batch file. Do not use `Ctrl-C` or any other method to terminate `help_start.bat`.
   - On Linux, run the `help_end` script using the fully qualified path for the DB2 Information Center:

     ```
     <DB2 Information Center dir>/doc/bin/help_end
     ```

     Note: The help_end script contains the commands required to safely terminate the processes that were started with the help_start script. Do not use any other method to terminate the `help_start` script.
7. Restart the DB2 Information Center service.
   - On Windows, click **Start** → **Control Panel** → **Administrative Tools** → **Services**. Then right-click on **DB2 Information Center** service and select **Start**.
   - On Linux, enter the following command:

     ```
     /etc/init.d/db2icdv9 start
     ```

The updated DB2 Information Center displays the new and updated topics.

Related concepts:
- "DB2 Information Center installation options" in *Quick Beginnings for DB2 Servers*

Related tasks:
- "Installing the DB2 Information Center using the DB2 Setup wizard (Linux)" in *Quick Beginnings for DB2 Servers*
- "Installing the DB2 Information Center using the DB2 Setup wizard (Windows)" in *Quick Beginnings for DB2 Servers*

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 products. Lessons provide step-by-step instructions.

Before you begin:

You can view the XHTML version of the tutorial from the Information Center at http://publib.boulder.ibm.com/infocenter/db2help/.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials:

To view the tutorial, click on the title.

Native XML data store
Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

Visual Explain Tutorial
Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

Related concepts:

- "Visual Explain overview" in *Administration Guide: Implementation*

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 products.

DB2 documentation
Troubleshooting information can be found in the DB2 Troubleshooting Guide or the Support and Troubleshooting section of the DB2 Information Center. There you will find information on how to isolate and identify problems using DB2 diagnostic tools and utilities, solutions to some of the most common problems, and other advice on how to solve problems you might encounter with your DB2 products.

DB2 Technical Support Web site
Refer to the DB2 Technical Support Web site if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the DB2 Technical Support Web site at http://www.ibm.com/software/data/db2/udb/support.html

Related concepts:

- "Introduction to problem determination" in *Troubleshooting Guide*
- "Overview of the DB2 technical information" on page 349

Terms and Conditions

Appendix I. Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country/region or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product, and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks

Index

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/.

Printed in USA

SC10-4227-00

Spine information:

IBM DB2 **DB2 Version 9** **Data Movement Utilities** IBM