DB2

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Call Level Interface Guide and Reference, Volume 2

SC10-4225-00

DB2®

IBM

DB2 Version 9
for Linux, UNIX, and Windows

Call Level Interface Guide and Reference, Volume 2

SC10-4225-00

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

Edition Notice

You can order IBM publications online or through your local IBM representative.

- To order publications online, go to the IBM Publications Center at www.ibm.com/shop/publications/order
- To find your local IBM representative, go to the IBM Directory of Worldwide Contacts at www.ibm.com/planetwide

To order DB2 publications from DB2 Marketing and Sales in the United States or Canada, call 1-800-IBM-4YOU (426-4968).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. DB2 CLI functions

This chapter provides a description of each DB2 CLI function. The function summary presents the functions organized by category. The function listing describes each function in detail.

CLI and ODBC function summary

Depr in the ODBC column indicates that the function has been deprecated in ODBC.

The SQL/CLI column can have the following values:
- 95 - Defined in the SQL/CLI 9075-3 specification.
- SQL3 - Defined in the SQL/CLI part of the ISO SQL3 draft replacement for SQL/CLI 9075-3.

Table 1. DB2 CLI Function list by category

Task Function Name	ODBC 3.0	SQL/CLI	DB2 CLI First Version Supported	Purpose
Connecting to a data source				
`SQLAllocConnect()`	Depr	95	V 1.1	Obtains a connection handle.
`SQLAllocEnv()`	Depr	95	V 1.1	Obtains an environment handle. One environment handle is used for one or more connections.
`SQLAllocHandle()`	Core	95	V 5	Obtains a handle.
`SQLBrowseConnect()`	Level 1	95	V 5	Get required attributes to connect to a data source.
`SQLConnect()`	Core	95	V 1.1	Connects to specific driver by data source name, user Id, and password.
`SQLDriverConnect()`	Core	SQL3	V 2.1 [a]	Connects to a specific driver by connection string or optionally requests that the Driver Manager and driver display connection dialogs for the user. **Note:** This function is also affected by the additional IBM® keywords supported in the ODBC.INI file.
`SQLDrivers()`	Core	No	NONE	DB2 CLI does not support this function as it is implemented by a Driver Manager.
`SQLSetConnectAttr()`	Core	95	V 5	Set connection attributes.
`SQLSetConnectOption()`	Depr	95	V 2.1	Set connection attributes.
`SQLSetConnection()`	No	SQL3	V 2.1	Sets the current active connection. This function only needs to be used when using embedded SQL within a DB2 CLI application with multiple concurrent connections.
DataLink functions				
`SQLBuildDataLink()`	No	Yes	V 5.2	Build DATALINK Value

Table 1. DB2 CLI Function list by category (continued)

Task Function Name	ODBC 3.0	SQL/CLI	DB2 CLI First Version Supported	Purpose
SQLGetDataLinkAttr()	No	Yes	V 5.2	Get DataLink attribute value
Obtaining information about a driver and data source				
SQLDataSources()	Lvl 2	95	V 1.1	Returns the list of available data sources.
SQLGetInfo()	Core	95	V 1.1	Returns information about a specific driver and data source.
SQLGetFunctions()	Core	95	V 1.1	Returns supported driver functions.
SQLGetTypeInfo()	Core	95	V 1.1	Returns information about supported data types.
Setting and retrieving driver options				
SQLSetEnvAttr()	Core	95	V 2.1	Sets an environment option.
SQLGetEnvAttr()	Core	95	V 2.1	Returns the value of an environment option.
SQLGetConnectAttr()	Lvl 1	95	V 5	Returns the value of a connection option.
SQLGetConnectOption()	Depr	95	V 2.1 [a]	Returns the value of a connection option.
SQLSetStmtAttr()	Core	95	V 5	Sets a statement attribute.
SQLSetStmtOption()	Depr	95	V 2.1 [a]	Sets a statement option.
SQLGetStmtAttr()	Core	95	V 5	Returns the value of a statement attribute.
SQLGetStmtOption()	Depr	95	V 2.1 [a]	Returns the value of a statement option.
Preparing SQL requests				
SQLAllocStmt()	Depr	95	V 1.1	Allocates a statement handle.
SQLPrepare()	Core	95	V 1.1	Prepares an SQL statement for later execution.
SQLExtendedPrepare()	No	No	V 6	Prepares an array of statement attributes for an SQL statement for later execution.
SQLExtendedBind()	No	No	V 6	Bind an array of columns instead of using repeated calls to SQLBindCol() and SQLBindParameter()
SQLBindParameter()	Lvl 1	95 [b]	V 2.1	Assigns storage for a parameter in an SQL statement (ODBC 2.0)
SQLSetParam()	Depr	No	V 1.1	Assigns storage for a parameter in an SQL statement (ODBC 1.0). **Note:** In ODBC 2.0 this function has been replaced by SQLBindParameter().
SQLParamOptions()	Depr	No	V 2.1	Specifies the use of multiple values for parameters.
SQLGetCursorName()	Core	95	V 1.1	Returns the cursor name associated with a statement handle.
SQLSetCursorName()	Core	95	V 1.1	Specifies a cursor name.
Submitting requests				
SQLDescribeParam()	Level 2	SQL3	V 5	Returns description of a parameter marker.
SQLExecute()	Core	95	V 1.1	Executes a prepared statement.

Table 1. DB2 CLI Function list by category (continued)

Task Function Name	ODBC 3.0	SQL/CLI	DB2 CLI First Version Supported	Purpose
SQLExecDirect()	Core	95	V 1.1	Executes a statement.
SQLNativeSql()	Lvl 2	95	V 2.1 [a]	Returns the text of an SQL statement as translated by the driver.
SQLNumParams()	Lvl 2	95	V 2.1 [a]	Returns the number of parameters in a statement.
SQLParamData()	Lvl 1	95	V 2.1 [a]	Used in conjunction with SQLPutData() to supply parameter data at execution time. (Useful for long data values.)
SQLPutData()	Core	95	V 2.1 [a]	Send part or all of a data value for a parameter. (Useful for long data values.)
Retrieving results and information about results				
SQLRowCount()	Core	95	V 1.1	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols()	Core	95	V 1.1	Returns the number of columns in the result set.
SQLDescribeCol()	Core	95	V 1.1	Describes a column in the result set.
SQLColAttribute()	Core	Yes	V 5	Describes attributes of a column in the result set.
SQLColAttributes()	Depr	Yes	V 1.1	Describes attributes of a column in the result set.
SQLColumnPrivileges()	Level 2	95	V 2.1	Get privileges associated with the columns of a table.
SQLSetColAttributes()	No	No	V 2.1	Sets attributes of a column in the result set.
SQLBindCol()	Core	95	V 1.1	Assigns storage for a result column and specifies the data type.
SQLFetch()	Core	95	V 1.1	Returns a result row.
SQLFetchScroll()	Core	95	V 5	Returns a rowset from a result row.
SQLExtendedFetch()	Depr	95	V 2.1	Returns multiple result rows.
SQLGetData()	Core	95	V 1.1	Returns part or all of one column of one row of a result set. (Useful for long data values.)
SQLMoreResults()	Lvl 1	SQL3	V 2.1 [a]	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLNextResult()	No	Yes	V7.1	SQLNextResult allows non-sequential access to multiple result sets returned from a stored procedure.
SQLError()	Depr	95	V 1.1	Returns additional error or status information.
SQLGetDiagField()	Core	95	V 5	Get a field of diagnostic data.
SQLGetDiagRec()	Core	95	V 5	Get multiple fields of diagnostic data.
SQLSetPos()	Level 1	SQL3	V 5	Set the cursor position in a rowset.
SQLGetSQLCA()	No	No	V 2.1	Returns the SQLCA associated with a statement handle.

Table 1. DB2 CLI Function list by category (continued)

Task Function Name	ODBC 3.0	SQL/CLI	DB2 CLI First Version Supported	Purpose
SQLBulkOperations()	Level 1	No	V 6	Perform bulk insertions, updates, deletions, and fetches by bookmark.
Descriptors				
SQLCopyDesc()	Core	95	V 5	Copy descriptor information between handles.
SQLGetDescField()	Core	95	V 5	Get single field settings of a descriptor record.
SQLGetDescRec()	Core	95	V 5	Get multiple field settings of a descriptor record.
SQLSetDescField()	Core	95	V 5	Set a single field of a descriptor record.
SQLSetDescRec()	Core	95	V 5	Set multiple field settings of a descriptor record.
Large object support				
SQLBindFileToCol()	No	No	V 2.1	Associates LOB file reference with a LOB column.
SQLBindFileToParam()	No	No	V 2.1	Associates LOB file reference with a parameter marker.
SQLGetLength()	No	SQL3	V 2.1	Gets length of a string referenced by a LOB locator.
SQLGetPosition()	No	SQL3	V 2.1	Gets the position of a string within a source string referenced by a LOB locator.
SQLGetSubString()	No	SQL3	V 2.1	Creates a new LOB locator that references a substring within a source string (the source string is also represented by a LOB locator).
Obtaining information about the data source's system tables (catalog functions)				
SQLColumns()	Lvl 1	SQL3	V 2.1 [a]	Returns the list of column names in specified tables.
SQLForeignKeys()	Lvl 2	SQL3	V 2.1	Returns a list of column names that comprise foreign keys, if they exist for a specified table.
SQLPrimaryKeys()	Lvl 1	SQL3	V 2.1	Returns the list of column name(s) that comprise the primary key for a table.
SQLProcedureColumns()	Lvl 2	No	V 2.1	Returns the list of input and output parameters for the specified procedures.
SQLProcedures()	Lvl 2	No	V 2.1	Returns the list of procedure names stored in a specific data source.
SQLSpecialColumns()	Core	SQL3	V 2.1 [a]	Returns information about the optimal set of columns that uniquely identifies a row in a specified table.
SQLStatistics()	Core	SQL3	V 2.1 [a]	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges()	Lvl 2	SQL3	V 2.1	Returns a list of tables and the privileges associated with each table.

Table 1. DB2 CLI Function list by category (continued)

Task Function Name	ODBC 3.0	SQL/CLI	DB2 CLI First Version Supported	Purpose
`SQLTables()`	Core	SQL3	V 2.1 [a]	Returns the list of table names stored in a specific data source.
Terminating a statement				
`SQLFreeHandle()`	Core	95	V 1.1	Free handle resources.
`SQLFreeStmt()`	Core	95	V 1.1	End statement processing and closes the associated cursor, discards pending results, and, optionally, frees all resources associated with the statement handle.
`SQLCancel()`	Core	95	V 1.1	Cancels an SQL statement.
`SQLTransact()`	Depr	No	V 1.1	Commits or rolls back a transaction.
`SQLCloseCursor()`	Core	95	V 5	Commits or rolls back a transaction.
Terminating a connection				
`SQLDisconnect()`	Core	95	V 1.1	Closes the connection.
`SQLEndTran()`	Core	95	V 5	Ends transaction of a connection.
`SQLFreeConnect()`	Depr	95	V 1.1	Releases the connection handle.
`SQLFreeEnv()`	Depr	95	V 1.1	Releases the environment handle.

Note:

[a] Runtime support for this function was also available in the DB2® Client Application Enabler for DOS Version 1.2 product.

[b] `SQLBindParam()` has been replaced by `SQLBindParameter()`.

The ODBC function(s):

- SQLSetScrollOptions() is supported for runtime only, because it has been superceded by the SQL_CURSOR_TYPE, SQL_CONCURRENCY, SQL_KEYSET_SIZE, and SQL_ROWSET_SIZE statement options.
- SQLDrivers() is implemented by the ODBC driver manager.

Related concepts:

- "Initialization and termination in CLI overview" in *Call Level Interface Guide and Reference, Volume 1*
- "Introduction to DB2 CLI and ODBC" in *Call Level Interface Guide and Reference, Volume 1*
- "Transaction processing in CLI overview" in *Call Level Interface Guide and Reference, Volume 1*

SQLAllocConnect function (CLI) - Allocate connection handle

Deprecated:

Note:

In ODBC 3.0, `SQLAllocConnect()` has been deprecated and replaced with `SQLAllocHandle()`.

Although this version of DB2 CLI continues to support SQLAllocConnect(), it is recommended that you use SQLAllocHandle() in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLAllocConnect(henv, &hdbc);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6

SQLAllocEnv function (CLI) - Allocate environment handle

Deprecated:

Note:

In ODBC 3.0, SQLAllocEnv() has been deprecated and replaced with SQLAllocHandle().

Although this version of DB2 CLI continues to support SQLAllocEnv(), we recommend that you use SQLAllocHandle() in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLAllocEnv(&henv);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6

SQLAllocHandle function (CLI) - Allocate handle

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

SQLAllocHandle() is a generic function that allocates environment, connection, statement, or descriptor handles.

Note: This function replaces the deprecated ODBC 2.0 functions SQLAllocConnect(), SQLAllocEnv(), and SQLAllocStmt().

Syntax:

```
SQLRETURN   SQLAllocHandle (
               SQLSMALLINT      HandleType,         /* fHandleType */
               SQLHANDLE        InputHandle,        /* hInput */
               SQLHANDLE        *OutputHandlePtr);  /* *phOutput */
```

Function Arguments:

Table 2. SQLAllocHandle arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	The type of handle to be allocated by SQLAllocHandle(). Must be one of the following values: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
SQLHANDLE	*InputHandle*	input	Existing handle to use as a context for the new handle being allocated. If HandleType is SQL_HANDLE_ENV, this is SQL_NULL_HANDLE. If HandleType is SQL_HANDLE_DBC, this must be an environment handle, and if it is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle.
SQLHANDLE *	*OutputHandlePtr*	output	Pointer to a buffer in which to return the handle to the newly allocated data structure.

Usage:

SQLAllocHandle() is used to allocate environment, connection, statement, and descriptor handles. An application can allocate multiple environment, connection, statement, or descriptor handles at any time a valid *InputHandle* exists.

If the application calls SQLAllocHandle() with **OutputHandlePtr* set to an existing environment, connection, statement, or descriptor handle, DB2 CLI overwrites the handle, and new resources appropriate to the handle type are allocated. There are no changes made to the CLI resources associated with the original handle.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

If SQLAllocHandle() returns SQL_INVALID_HANDLE, it will set *OutputHandlePtr* to SQL_NULL_HENV, SQL_NULL_HDBC, SQL_NULL_HSTMT, or SQL_NULL_HDESC, depending on the value of *HandleType*, unless the output argument is a null pointer. The application can then obtain additional information from the diagnostic data structure associated with the handle in the *InputHandle* argument.

Diagnostics:

Table 3. SQLAllocHandle SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	The HandleType argument was SQL_HANDLE_STMT or SQL_HANDLE_DESC, but the connection handle specified by the *InputHandle* argument did not have an open connection. The connection process must be completed successfully (and the connection must be open) for DB2 CLI to allocate a statement or descriptor handle.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	The *HandleType* argument was SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC; and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY014	No more handles.	The limit for the number of handles that can be allocated for the type of handle indicated by the *HandleType* argument has been reached, or in some cases, insufficient system resources exist to properly initialize the new handle.
HY092	Option type out of range.	The *HandleType* argument was not one of: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC

Restrictions:

None.

Example:

```
SQLHANDLE henv; /* environment handle */
SQLHANDLE hdbc; /* connection handle */
SQLHANDLE hstmt; /* statement handle */
SQLHANDLE hdesc; /* descriptor handle */

/* ... */

/* allocate an environment handle */
cliRC = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

/* ... */

/* allocate a database connection handle */
cliRC = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

/* ... */
/* connect to database using hdbc */
/* ... */

/* allocate one or more statement handles */
```

```
cliRC = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

/* ... */
/* allocate a descriptor handle */
cliRC = SQLAllocHandle(SQL_HANDLE_DESC, hstmt, &hdesc);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Allocating statement handles in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Initializing CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLFreeHandle function (CLI) - Free handle resources" on page 140
- "SQLSetEnvAttr function (CLI) - Set environment attribute" on page 284

Related samples:

- "clihandl.c -- How to allocate and free handles"

SQLAllocStmt function (CLI) - Allocate a statement handle

Deprecated:

Note:

In ODBC 3.0, `SQLAllocStmt()` has been deprecated and replaced with `SQLAllocHandle()`.

Although this version of DB2 CLI continues to support `SQLAllocStmt()`, we recommend that you use `SQLAllocHandle()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLAllocStmt(hdbc, &hstmt);
```

for example, would be rewritten using the new function as:

```
SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6

SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLBindCol() is used to associate (bind) columns in a result set to either:

- Application variables or arrays of application variables (storage buffers), for all C data types. Data is transferred from the DBMS to the application when SQLFetch() or SQLFetchScroll() is called. Data conversion might occur as the data is transferred.
- A LOB locator, for LOB columns. A LOB locator, not the data itself, is transferred from the DBMS to the application when SQLFetch() is called.

  Alternatively, LOB columns can be bound directly to a file using SQLBindFileToCol().

SQLBindCol() is called once for each column in the result set that the application needs to retrieve.

In general, SQLPrepare(), SQLExecDirect() or one of the schema functions is called before this function, and SQLFetch(), SQLFetchScroll(), SQLBulkOperations(), or SQLSetPos() is called after. Column attributes might also be needed before calling SQLBindCol(), and can be obtained using SQLDescribeCol() or SQLColAttribute().

Syntax:

```
SQLRETURN   SQLBindCol (
               SQLHSTMT          StatementHandle,          /* hstmt */
               SQLUSMALLINT      ColumnNumber,             /* icol */
               SQLSMALLINT       TargetType,               /* fCType */
               SQLPOINTER        TargetValuePtr,           /* rgbValue */
               SQLLEN            BufferLength,             /* dbValueMax */
               SQLLEN            *StrLen_or_IndPtr);       /* *pcbValue */
```

Function arguments:

Table 4. SQLBindCol arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLUSMALLINT	*ColumnNumber*	input	Number identifying the column. Columns are numbered sequentially, from left to right. • Column numbers start at 1 if bookmarks are not used (SQL_ATTR_USE_BOOKMARKS statement attribute set to SQL_UB_OFF). • Column numbers start at 0 if bookmarks are used (the statement attribute is set to SQL_UB_ON). Column 0 is the bookmark column.

Table 4. SQLBindCol arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*TargetType*	input	The C data type for column number *ColumnNumber* in the result set. When the application retrieves data from the data source, it will convert the data to this C type. When using `SQLBulkOperations()` or `SQLSetPos()`, the driver will convert data from this C data type when sending information to the data source. The following types are supported: • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DECIMAL_IBM • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_NUMERIC [a] • SQL_C_SBIGINT • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TINYINT • SQL_C_UBIGINT • SQL_C_UTINYINT • SQL_C_WCHAR Specifying SQL_C_DEFAULT causes data to be transferred to its default C data type.
SQLPOINTER	*TargetValuePtr*	input/output (deferred)	Pointer to buffer or an array of buffers with either column-wise or row-wise binding, where DB2 CLI is to store the column data or the LOB locator when the fetch occurs. This buffer is used to return data when any of the following functions are called: `SQLFetch()`, `SQLFetchScroll()`, `SQLSetPos()` using the *Operation* argument SQL_REFRESH, or `SQLBulkOperations()` using the *Operation* argument SQL_FETCH_BY_BOOKMARK. Otherwise, `SQLBulkOperations()` and `SQLSetPos()` use the buffer to retrieve data. If *TargetValuePtr* is null, the column is unbound. All columns can be unbound with a call to `SQLFreeStmt()` with the SQL_UNBIND option.

Table 4. SQLBindCol arguments (continued)

Data type	Argument	Use	Description
SQLLEN	*BufferLength*	input	Size in bytes of *TargetValuePtr* buffer available to store the column data or the LOB locator. If *TargetType* denotes a binary or character string (either single or double byte) or is SQL_C_DEFAULT for a column returning variable length data, then *BufferLength* must be > 0, or an error will be returned. Note that for character data, the driver counts the NULL termination character and so space must be allocated for it. For all other data types, this argument is ignored.
SQLLEN *	*StrLen_or_IndPtr*	input/output (deferred)	Pointer to value (or array of values) which indicates the number of bytes DB2 CLI has available to return in the *TargetValuePtr* buffer. If *TargetType* is a LOB locator, the size of the locator is returned, not the size of the LOB data. This buffer is used to return data when any of the following functions are called: `SQLFetch()`, `SQLFetchScroll()`, `SQLSetPos()` using the *Operation* argument SQL_REFRESH, or `SQLBulkOperations()` using the *Operation* argument SQL_FETCH_BY_BOOKMARK. Otherwise, `SQLBulkOperations()` and `SQLSetPos()` use the buffer to retrieve data. `SQLFetch()` returns SQL_NULL_DATA in this argument if the data value of the column is null. This pointer value must be unique for each bound column, or NULL. A value of SQL_COLUMN_IGNORE, SQL_NTS, SQL_NULL_DATA, or the length of the data can be set for use with `SQLBulkOperations()`. SQL_NO_LENGTH might also be returned, refer to the Usage section below for more information.

- For this function, both *TargetValuePtr* and *StrLen_or_IndPtr* are deferred outputs, meaning that the storage locations these pointers point to do not get updated until a result set row is fetched. As a result, the locations referenced by these pointers must remain valid until `SQLFetch()` or `SQLFetchScroll()` is called. For example, if `SQLBindCol()` is called within a local function, `SQLFetch()` must be called from within the same scope of the function or the *TargetValuePtr* buffer must be allocated as static or global.
- DB2 CLI will be able to optimize data retrieval for all variable length data types if *TargetValuePtr* is placed consecutively in memory after *StrLen_or_IndPtr*.

Usage:

Call `SQLBindCol()` once for each column in the result set for which either the data or, for LOB columns, the LOB locator is to be retrieved. When `SQLFetch()` or `SQLFetchScroll()` is called to retrieve data from the result set, the data in each of the bound columns is placed in the locations assigned by the *TargetValuePtr* and *StrLen_or_IndPtr* pointers. When the statement attribute SQL_ATTR_ROW_ARRAY_SIZE is greater than 1, then *TargetType* should refer to

an array of buffers. If *TargetType* is a LOB locator, a locator value is returned, not the actual LOB data. The LOB locator references the entire data value in the LOB column.

Columns are identified by a number, assigned sequentially from left to right.
- Column numbers start at 1 if bookmarks are not used (SQL_ATTR_USE_BOOKMARKS statement attribute set to SQL_UB_OFF).
- Column numbers start at 0 if bookmarks are used (the statement attribute set to SQL_UB_ON).

After columns have been bound, in subsequent fetches the application can change the binding of these columns or bind previously unbound columns by calling `SQLBindCol()`. The new binding does not apply to data already fetched, it will be used on the next fetch. To unbind a single column (including columns bound with `SQLBindFileToCol()`), call `SQLBindCol()` with the *TargetValuePtr* pointer set to NULL. To unbind all the columns, the application should call `SQLFreeStmt()` with the *Option* input set to SQL_UNBIND.

The application must ensure enough storage is allocated for the data to be retrieved. If the buffer is to contain variable length data, the application must allocate as much storage as the maximum length of the bound column plus the NULL terminator. Otherwise, the data might be truncated. If the buffer is to contain fixed length data, DB2 CLI assumes the size of the buffer is the length of the C data type. If data conversion is specified, the required size might be affected.

If string truncation does occur, SQL_SUCCESS_WITH_INFO is returned and *StrLen_or_IndPtr* will be set to the actual size of *TargetValuePtr* available for return to the application.

Truncation is also affected by the SQL_ATTR_MAX_LENGTH statement attribute (used to limit the amount of data returned to the application). The application can specify not to report truncation by calling `SQLSetStmtAttr()` with SQL_ATTR_MAX_LENGTH and a value for the maximum length to return for all variable length columns, and by allocating a *TargetValuePtr* buffer of the same size (plus the null-terminator). If the column data is larger than the set maximum length, SQL_SUCCESS will be returned when the value is fetched and the maximum length, not the actual length, will be returned in *StrLen_or_IndPtr*.

If the column to be bound is a SQL_GRAPHIC, SQL_VARGRAPHIC or SQL_LONGVARGRAPHIC type, then *TargetType* can be set to SQL_C_DBCHAR or SQL_C_CHAR. If *TargetType* is SQL_C_DBCHAR, the data fetched into the *TargetValuePtr* buffer will be null-terminated with a double byte null-terminator. If *TargetType* is SQL_C_CHAR, then there will be no null-termination of the data. In both cases, the length of the *TargetValuePtr* buffer (*BufferLength*) is in units of bytes and should therefore be a multiple of 2. It is also possible to force DB2 CLI to null terminate graphic strings using the PATCH1 keyword.

Note: SQL_NO_TOTAL will be returned in *StrLen_or_IndPtr* if:
- The SQL type is a variable length type, and
- *StrLen_or_IndPtr* and *TargetValuePtr* are contiguous, and
- The column type is NOT NULLABLE, and
- String truncation occurred.

Descriptors and SQLBindCol

The following sections describe how `SQLBindCol()` interacts with descriptors.

Note: Calling SQLBindCol() for one statement can affect other statements. This occurs when the ARD associated with the statement is explicitly allocated and is also associated with other statements. Because SQLBindCol() modifies the descriptor, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate this descriptor from the other statements before calling SQLBindCol().

Argument mappings

Conceptually, SQLBindCol() performs the following steps in sequence:
- Calls SQLGetStmtAttr() to obtain the ARD handle.
- Calls SQLGetDescField() to get this descriptor's SQL_DESC_COUNT field, and if the value in the *ColumnNumber* argument exceeds the value of SQL_DESC_COUNT, calls SQLSetDescField() to increase the value of SQL_DESC_COUNT to *ColumnNumber*.
- Calls SQLSetDescField() multiple times to assign values to the following fields of the ARD:
  - Sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of *TargetType*.
  - Sets one or more of SQL_DESC_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE as appropriate for *TargetType*.
  - Sets the SQL_DESC_OCTET_LENGTH field to the value of *BufferLength*.
  - Sets the SQL_DESC_DATA_PTR field to the value of *TargetValue*.
  - Sets the SQL_DESC_INDICATOR_PTR field to the value of *StrLen_or_IndPtr* (see the following paragraph).
  - Sets the SQL_DESC_OCTET_LENGTH_PTR field to the value of *StrLen_or_IndPtr* (see the following paragraph).

The variable that the *StrLen_or_IndPtr* argument refers to is used for both indicator and length information. If a fetch encounters a null value for the column, it stores SQL_NULL_DATA in this variable; otherwise, it stores the data length in this variable. Passing a null pointer as *StrLen_or_IndPtr* keeps the fetch operation from returning the data length, but makes the fetch fail if it encounters a null value and has no way to return SQL_NULL_DATA.

If the call to SQLBindCol() fails, the content of the descriptor fields it would have set in the ARD are undefined, and the value of the SQL_DESC_COUNT field of the ARD is unchanged.

Implicit resetting of COUNT field

SQLBindCol() sets SQL_DESC_COUNT to the value of the *ColumnNumber* argument only when this would increase the value of SQL_DESC_COUNT. If the value in the *TargetValuePtr* argument is a null pointer and the value in the *ColumnNumber* argument is equal to SQL_DESC_COUNT (that is, when unbinding the highest bound column), then SQL_DESC_COUNT is set to the number of the highest remaining bound column.

Cautions regarding SQL_C_DEFAULT

To retrieve column data successfully, the application must determine correctly the length and starting point of the data in the application buffer. When the application specifies an explicit *TargetType*, application misconceptions are readily detected. However, when the application specifies a *TargetType* of SQL_C_DEFAULT, SQLBindCol() can be applied to a column of a different data

type from the one intended by the application, either from changes to the metadata or by applying the code to a different column. In this case, the application might fail to determine the start or length of the fetched column data. This can lead to unreported data errors or memory violations.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 5. SQLBindCol SQLSTATEs

SQLSTATE	Description	Explanation
07009	Invalid descriptor index	The value specified for the argument *ColumnNumber* exceeded the maximum number of columns in the result set, or the value specified was less than 0.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	*TargetType* was not a valid data type or SQL_C_DEFAULT.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than 1 and the argument *TargetType* is either SQL_C_CHAR, SQL_C_BINARY or SQL_C_DEFAULT.
HYC00	Driver not capable.	DB2 CLI recognizes, but does not support the data type specified in the argument *TargetType* A LOB locator C data type was specified, but the connected server does not support LOB data types.

Note: Additional diagnostic messages relating to the bound columns might be reported at fetch time.

Restrictions:

The LOB data support is only available when connected to a server that supports large object data types. If the application attempts to specify a LOB locator C data type for a server that does not support it, SQLSTATE **HY**C00 will be returned.

Example:

```
/* bind column 1 to variable */
cliRC = SQLBindCol(hstmt, 1, SQL_C_SHORT, &deptnumb.val, 0, &deptnumb.ind);
```

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "LOB locators in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Retrieving array data in CLI applications using column-wise binding" in *Call Level Interface Guide and Reference, Volume 1*
- "Retrieving array data in CLI applications using row-wise binding" in *Call Level Interface Guide and Reference, Volume 1*
- "Retrieving query results in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "C data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*
- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column" on page 16
- "SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows" on page 43
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "SQLSetPos function (CLI) - Set the cursor position in a rowset" on page 287

Related samples:

- "tbinfo.c -- How to get information about tables from the system catalog tables"

SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column

Purpose:

Specification:	**DB2 CLI** 2.1		

`SQLBindFileToCol()` is used to associate or bind a LOB or XML column in a result set to a file reference or an array of file references. This enables data in that column to be transferred directly into a file when each row is fetched for the statement handle.

The LOB file reference arguments (file name, file name length, file reference options) refer to a file within the application's environment (on the client). Before fetching each row, the application must make sure that these variables contain the

name of a file, the length of the file name, and a file option (new / overwrite / append). These values can be changed between each row fetch operation.

Syntax:

```
SQLRETURN   SQLBindFileToCol (SQLHSTMT          StatementHandle,   /* hstmt */
                              SQLUSMALLINT      ColumnNumber,      /* icol */
                              SQLCHAR           *FileName,
                              SQLSMALLINT       *FileNameLength,
                              SQLUINTEGER       *FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER        *StringLength,
                              SQLINTEGER        *IndicatorValue);
```

Function arguments:

Table 6. SQLBindFileToCol arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*icol*	input	Number identifying the column. Columns are numbered sequentially, from left to right, starting at 1.
SQLCHAR *	*FileName*	input (deferred)	Pointer to the location that will contain the file name or an array of file names at the time of the next fetch using the *StatementHandle*. This is either the complete path name of the file(s) or a relative file name(s). If relative file name(s) are provided, they are appended to the current path of the running application. This pointer cannot be NULL.
SQLSMALLINT *	*FileNameLength*	input (deferred)	Pointer to the location that will contain the length of the file name (or an array of lengths) at the time of the next fetch using the *StatementHandle*. If this pointer is NULL, then the *FileName* will be considered a null-terminated string, similar to passing a length of SQL_NTS. The maximum value of the file name length is 255.
SQLUINTEGER *	*FileOptions*	input (deferred)	Pointer to the location that will contain the file option or (array of file options) to be used when writing the file at the time of the next fetch using the *StatementHandle*. The following *FileOptions* are supported: **SQL_FILE_CREATE** Create a new file. If a file by this name already exists, SQL_ERROR will be returned. **SQL_FILE_OVERWRITE** If the file already exists, overwrite it. Otherwise, create a new file. **SQL_FILE_APPEND** If the file already exists, append the data to it. Otherwise, create a new file. Only one option can be chosen per file, there is no default.
SQLSMALLINT	*MaxFileNameLength*	input	This specifies the length of the *FileName* buffer or, if the application uses `SQLFetchScroll()` to retrieve multiple rows for the LOB column, this specifies the length of each element in the *FileName* array.

Table 6. SQLBindFileToCol arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	*StringLength*	output (deferred)	Pointer to the location that contains the length (or array of lengths) in bytes of the LOB data that is returned. If this pointer is NULL, nothing is returned.
SQLINTEGER *	*IndicatorValue*	output (deferred)	Pointer to the location that contains an indicator value (or array of values).

Usage:

The application calls `SQLBindFileToCol()` once for each column that should be transferred directly to a file when a row is fetched. LOB data is written directly to the file without any data conversion, and without appending null-terminators. XML data is written out in UTF-8, with an XML declaration generated according to the setting of the SQL_ATTR_XML_DECLARATION connection or statement attribute.

FileName, *FileNameLength*, and *FileOptions* must be set before each fetch. When `SQLFetch()` or `SQLFetchScroll()` is called, the data for any column which has been bound to a LOB file reference is written to the file or files pointed to by that file reference. Errors associated with the deferred input argument values of `SQLBindFileToCol()` are reported at fetch time. The LOB file reference, and the deferred *StringLength* and *IndicatorValue* output arguments are updated between fetch operations.

If `SQLFetchScroll()` is used to retrieve multiple rows for the LOB column, *FileName, FileNameLength,* and *FileOptions* point to arrays of LOB file reference variables. In this case, *MaxFileNameLength* specifies the length of each element in the *FileName* array and is used by DB2 CLI to determine the location of each element in the *FileName* array. The contents of the array of file references must be valid at the time of the `SQLFetchScroll()` call. The *StringLength* and *IndicatorValue* pointers each point to an array whose elements are updated upon the `SQLFetchScroll()` call.

Using `SQLFetchScroll()`, multiple LOB values can be written to multiple files, or to the same file depending on the file names specified. If writing to the same file, the SQL_FILE_APPEND file option should be specified for each file name entry. Only column-wise binding of arrays of file references is supported with SQLFetchScroll().

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 7. SQLBindFileToCol SQLSTATEs

SQLSTATE	Description	Explanation
07009	Invalid column number.	The value specified for the argument *icol* was less than 1. The value specified for the argument *icol* exceeded the maximum number of columns supported by the data source.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	*FileName*, *StringLength* or *FileOptions* is a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument *MaxFileNameLength* was less than 0.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.

Restrictions:

This function is not available when connected to DB2 servers that do not support large object data types. Call `SQLGetFunctions()` with the function type set to SQL_API_SQLBINDFILETOCOL and check the *SupportedPtr* output argument to determine if the function is supported for the current connection.

Example:

```
/* bind a file to the BLOB column */
  rc = SQLBindFileToCol(hstmt,
                        1,
                        fileName,
                        &fileNameLength,
                        &fileOption,
                        14,
                        NULL,
                        &fileInd);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Large object usage in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Retrieving query results in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126

Related samples:

- "dtlob.c -- How to read and write LOB data"

SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter

Purpose:

Specification:	**DB2 CLI** 2.1		

`SQLBindFileToParam()` is used to associate or bind a parameter marker in an SQL statement to a file reference or an array of file references. This enables data from the file to be transferred directly into a LOB or XML column when the statement is subsequently executed.

The LOB file reference arguments (file name, file name length, file reference options) refer to a file within the application's environment (on the client). Before calling `SQLExecute()` or `SQLExecDirect()`, the application must make sure that this information is available in the deferred input buffers. These values can be changed between `SQLExecute()` calls.

Syntax:

```
SQLRETURN SQLBindFileToParam (
             SQLHSTMT          StatementHandle,          /* hstmt */
             SQLUSMALLINT      TargetType,               /* ipar */
             SQLSMALLINT       DataType,                 /* fSqlType */
             SQLCHAR           *FileName,
             SQLSMALLINT       *FileNameLength,
             SQLUINTEGER       *FileOptions,
             SQLSMALLINT       MaxFileNameLength,
             SQLINTEGER        *IndicatorValue);
```

Function arguments:

Table 8. SQLBindFileToParam arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*TargetType*	input	Parameter marker number. Parameters are numbered sequentially, from left to right, starting at 1.

Table 8. SQLBindFileToParam arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*DataType*	input	SQL Data Type of the column. The data type must be one of: • SQL_BLOB • SQL_CLOB • SQL_DBCLOB • SQL_XML
SQLCHAR *	*FileName*	input (deferred)	Pointer to the location that will contain the file name or an array of file names when the statement (*StatementHandle*) is executed. This is either the complete path name of the file or a relative file name. If a relative file name is provided, it is appended to the current path of the client process. This argument cannot be NULL.
SQLSMALLINT *	*FileNameLength*	input (deferred)	Pointer to the location that will contain the length of the file name (or an array of lengths) at the time of the next `SQLExecute()` or `SQLExecDirect()` using the *StatementHandle*. If this pointer is NULL, then the *FileName* will be considered a null-terminated string, similar to passing a length of SQL_NTS. The maximum value of the file name length is 255.
SQLUINTEGER *	*FileOptions*	input (deferred)	Pointer to the location that will contain the file option (or an array of file options) to be used when reading the file. The location will be accessed when the statement (*StatementHandle*) is executed. Only one option is supported (and it must be specified): **SQL_FILE_READ** A regular file that can be opened, read and closed. (The length is computed when the file is opened) This pointer cannot be NULL.
SQLSMALLINT	*MaxFileNameLength*	input	This specifies the length of the *FileName* buffer. If the application calls `SQLParamOptions()` to specify multiple values for each parameter, this is the length of each element in the *FileName* array.
SQLINTEGER *	*IndicatorValue*	input (deferred)	Pointer to the location that contains an indicator value (or array of values), which is set to SQL_NULL_DATA if the data value of the parameter is to be null. It must be set to 0 (or the pointer can be set to null) when the data value is not null.

Usage:

The application calls `SQLBindFileToParam()` once for each parameter marker whose value should be obtained directly from a file when a statement is executed. Before the statement is executed, *FileName*, *FileNameLength*, and *FileOptions* values must be set. When the statement is executed, the data for any parameter which has been bound using `SQLBindFileToParam()` is read from the referenced file and passed to the server.

If the application uses SQLParamOptions() to specify multiple values for each parameter, then *FileName*, *FileNameLength*, and *FileOptions* point to an array of LOB file reference variables. In this case, *MaxFileNameLength* specifies the length of each element in the *FileName* array and is used by DB2 CLI to determine the location of each element in the *FileName* array.

A LOB parameter marker can be associated with (bound to) an input file using SQLBindFileToParam(), or with a stored buffer using SQLBindParameter(). The most recent bind parameter function call determines the type of binding that is in effect.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 9. SQLBindFileToParam SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY004	SQL data type out of range.	The value specified for *DataType* was not a valid SQL type for this function call.
HY009	Invalid argument value.	*FileName*, *FileOptions FileNameLength*, is a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the input argument *MaxFileNameLength* was less than 0.
HY093	Invalid parameter number.	The value specified for *TargetType* was either less than 1 or greater than the maximum number of parameters supported.
HYC00	Driver not capable.	The server does not support Large Object data types.

Restrictions:

This function is not available when connected to DB2 servers that do not support large object data types. Call SQLGetFunctions() with the function type set to SQL_API_SQLBINDFILETOPARAM and check the *SupportedPtr* output argument to determine if the function is supported for the current connection.

Example:

```
/* bind the file parameter */
rc = SQLBindFileToParam(hstmt,
                        3,
                        SQL_BLOB,
                        fileName,
                        &fileNameLength,
                        &fileOption,
                        14,
                        &fileInd);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Binding parameter markers in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLExecute function (CLI) - Execute a statement" on page 106
- "SQLParamOptions function (CLI) - Specify an input array for a parameter" on page 242
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dtlob.c -- How to read and write LOB data"

SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 2.0	

`SQLBindParameter()` is used to associate or bind parameter markers in an SQL statement to either:

- Application variables or arrays of application variables (storage buffers) for all C data types. In this case data is transferred from the application to the DBMS when `SQLExecute()` or `SQLExecDirect()` is called. Data conversion might occur as the data is transferred.
- A LOB locator, for SQL LOB data types. In this case a LOB locator value, not the LOB data itself, is transferred from the application to the server when the SQL statement is executed.

  Alternatively, LOB parameters can be bound directly to a file using `SQLBindFileToParam()`

This function must also be used to bind a parameter of a stored procedure CALL statement to the application where the parameter can be input, output or both.

SQLBindParameter

Syntax:

```
SQLRETURN  SQLBindParameter(
               SQLHSTMT          StatementHandle,     /* hstmt */
               SQLUSMALLINT      ParameterNumber,     /* ipar */
               SQLSMALLINT       InputOutputType,     /* fParamType */
               SQLSMALLINT       ValueType,           /* fCType */
               SQLSMALLINT       ParameterType,       /* fSqlType */
               SQLULEN           ColumnSize,          /* cbColDef */
               SQLSMALLINT       DecimalDigits,       /* ibScale */
               SQLPOINTER        ParameterValuePtr,   /* rgbValue */
               SQLLEN            BufferLength,        /* cbValueMax */
               SQLLEN            *StrLen_or_IndPtr);  /* pcbValue */
```

Function arguments:

Table 10. SQLBindParameter arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement Handle
SQLUSMALLINT	*ParameterNumber*	input	Parameter marker number, ordered sequentially left to right, starting at 1.
SQLSMALLINT	*InputOutputType*	input	The type of parameter. The value of the SQL_DESC_PARAMETER_TYPE field of the IPD is also set to this argument. The supported types are: • SQL_PARAM_INPUT: The parameter marker is associated with an SQL statement that is not a stored procedure CALL; or, it marks an input parameter of the CALLed stored procedure. When the statement is executed, the data for the parameter is sent to the server and as such, the *ParameterValuePtr* buffer must contain valid input data value(s), unless the *StrLen_or_IndPtr* buffer contains SQL_NULL_DATA or SQL_DATA_AT_EXEC (if the value should be sent via `SQLParamData()` and `SQLPutData()`). • SQL_PARAM_INPUT_OUTPUT: The parameter marker is associated with an input/output parameter of the CALLed stored procedure. When the statement is executed, the data for the parameter is sent to the server and as such, the *ParameterValuePtr* buffer must contain valid input data value(s), unless the *StrLen_or_IndPtr* buffer contains SQL_NULL_DATA or SQL_DATA_AT_EXEC (if the value should be sent via `SQLParamData()` and `SQLPutData()`). • SQL_PARAM_OUTPUT: The parameter marker is associated with an output parameter of the CALLed stored procedure or the return value of the stored procedure. After the statement is executed, data for the output parameter is returned to the application buffer specified by *ParameterValuePtr* and *StrLen_or_IndPtr*, unless both are NULL pointers, in which case the output data is discarded. If an output parameter does not have a return value then *StrLen_or_IndPtr* is set to SQL_NULL_DATA.

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*ValueType*	input	C data type of the parameter. The following types are supported: • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DECIMAL_IBM • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_NUMERIC [a] • SQL_C_SBIGINT • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TINYINT • SQL_C_UBIGINT • SQL_C_UTINYINT • SQL_C_WCHAR Specifying SQL_C_DEFAULT causes data to be transferred from its default C data type to the type indicated in *ParameterType*. **a** Windows® 32-bit only

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*ParameterType*	input	SQL data type of the parameter. The supported types are: • SQL_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONG • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SHORT • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR • SQL_XML **Note:** SQL_BLOB_LOCATOR, SQL_CLOB_LOCATOR, SQL_DBCLOB_LOCATOR are application related concepts and do not map to a data type for column definition during a CREATE TABLE statement.
SQLULEN	*ColumnSize*	input	Precision of the corresponding parameter marker. If *ParameterType* denotes: • A binary or single byte character string (for example, SQL_CHAR, SQL_BLOB), this is the maximum length in bytes for this parameter marker. • A double byte character string (for example, SQL_GRAPHIC), this is the maximum length in double-byte characters for this parameter. • SQL_DECIMAL, SQL_NUMERIC, this is the maximum decimal precision. • An XML value (SQL_XML) for an external routine argument, this is the maximum length in bytes, n, of the declared XML AS CLOB(n) argument. For all other parameters of type SQL_XML, this argument is ignored. • Otherwise, this argument is ignored.

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*DecimalDigits*	input	If *ParameterType* is SQL_DECIMAL or SQL_NUMERIC, *DecimalDigits* represents the scale of the corresponding parameter and sets the SQL_DESC_SCALE field of the IPD. If *ParameterType* is SQL_TYPE_TIMESTAMP or SQL_TYPE_TIME, *Decimal Digits* represents the precision of the corresponding parameter and sets the SQL_DESC_PRECISION field of the IPD. The precision of a time timestamp value is the number of digits to the right of the decimal point in the string representation of a time or timestamp (for example, the scale of yyyy-mm-dd hh:mm:ss.fff is 3). Other than for the *ParameterType* values mentioned here, *DecimalDigits* is ignored.
SQLPOINTER	*ParameterValuePtr*	input (deferred), output (deferred), or both	• On input (*InputOutputType* set to SQL_PARAM_INPUT, or SQL_PARAM_INPUT_OUTPUT): At execution time, if *StrLen_or_IndPtr* does not contain SQL_NULL_DATA or SQL_DATA_AT_EXEC, then *ParameterValuePtr* points to a buffer that contains the actual data for the parameter. If *StrLen_or_IndPtr* contains SQL_DATA_AT_EXEC, then *ParameterValuePtr* is an application-defined 32-bit value that is associated with this parameter. This 32-bit value is returned to the application via a subsequent `SQLParamData()` call. If `SQLParamOptions()` is called to specify multiple values for the parameter, then *ParameterValuePtr* is a pointer to a input buffer array of *BufferLength* bytes. • On output (*InputOutputType* set to SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT): *ParameterValuePtr* points to the buffer where the output parameter value of the stored procedure will be stored. If *InputOutputType* is set to SQL_PARAM_OUTPUT, and both *ParameterValuePtr* and *StrLen_or_IndPtr* are NULL pointers, then the output parameter value or the return value from the stored procedure call is discarded.

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLLEN	*BufferLength*	input	For character and binary data, *BufferLength* specifies the length of the *ParameterValuePtr* buffer (if is treated as a single element) or the length of each element in the *ParameterValuePtr* array (if the application calls `SQLParamOptions()` to specify multiple values for each parameter). For non-character and non-binary data, this argument is ignored -- the length of the *ParameterValuePtr* buffer (if it is a single element) or the length of each element in the *ParameterValuePtr* array (if `SQLParamOptions()` is used to specify an array of values for each parameter) is assumed to be the length associated with the C data type. For output parameters, *BufferLength* is used to determine whether to truncate character or binary output data in the following manner: • For character data, if the number of bytes available to return is greater than or equal to *BufferLength*, the data in *ParameterValuePtr* is truncated to *BufferLength-1* bytes and is null-terminated (unless null-termination has been turned off). • For binary data, if the number of bytes available to return is greater than *BufferLength*, the data in *ParameterValuePtr* is truncated to *BufferLength* bytes.

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLLEN *	*StrLen_or_IndPtr*	input (deferred), output (deferred), or both	If this is an input or input/output parameter: This is the pointer to the location which contains (when the statement is executed) the length of the parameter marker value stored at *ParameterValuePtr*. To specify a null value for a parameter marker, this storage location must contain SQL_NULL_DATA. If *ValueType* is SQL_C_CHAR, this storage location must contain either the exact length of the data stored at *ParameterValuePtr*, or SQL_NTS if the contents at *ParameterValuePtr* is null-terminated. If *ValueType* indicates character data (explicitly, or implicitly using SQL_C_DEFAULT), and this pointer is set to NULL, it is assumed that the application will always provide a null-terminated string in *ParameterValuePtr*. This also implies that this parameter marker will never have a null value. If *ParameterType* denotes a graphic data type and the *ValueType* is SQL_C_CHAR, the pointer to *StrLen_or_IndPtr* can never be NULL and the contents of *StrLen_or_IndPtr* can never hold SQL_NTS. In general for graphic data types, this length should be the number of octets that the double byte data occupies; therefore, the length should always be a multiple of 2. In fact, if the length is odd, then an error will occur when the statement is executed. When `SQLExecute()` or `SQLExecDirect()` is called, and *StrLen_or_IndPtr* points to a value of SQL_DATA_AT_EXEC, the data for the parameter will be sent with `SQLPutData()`. This parameter is referred to as a **data-at-execution** parameter.

Table 10. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	StrLen_or_IndPtr *(cont)*	input (deferred), output (deferred), or both	If `SQLSetStmtAttr()` is used with the SQL_ATTR_PARAMSET_SIZE attribute to specify multiple values for each parameter, *StrLen_or_IndPtr* points to an array of SQLINTEGER values where each of the elements can be the number of bytes in the corresponding *ParameterValuePtr* element (excluding the null-terminator), or SQL_NULL_DATA. If this is an output parameter (*InputOutputType* is set to SQL_PARAM_OUTPUT): This must be an output parameter or return value of a stored procedure CALL and points to one of the following, after the execution of the stored procedure: • number of bytes available to return in *ParameterValuePtr*, excluding the null-termination character. • SQL_NULL_DATA • SQL_NO_TOTAL if the number of bytes available to return cannot be determined.

Usage:

`SQLBindParameter()` extends the capability of the deprecated `SQLSetParam()` function, by providing a method of:

- Specifying whether a parameter is input, input / output, or output, necessary for proper handling of parameters for stored procedures.
- Specifying an array of input parameter values when `SQLSetStmtAttr()` with the SQL_ATTR_PARAMSET_SIZE attribute is used in conjunction with `SQLBindParameter()`.

This function can be called before `SQLPrepare()` if the data types and lengths of the target columns in the WHERE or UPDATE clause, or the parameters for the stored procedure are known. Otherwise, you can obtain the attributes of the target columns or stored procedure parameters after the statement is prepared using `SQLDescribeParam()`, and then bind the parameter markers.

Parameter markers are referenced by number (*ParameterNumber*) and are numbered sequentially from left to right, starting at 1.

The C buffer data type given by *ValueType* must be compatible with the SQL data type indicated by *ParameterType*, or an error will occur.

All parameters bound by this function remain in effect until one of the following occurs:

- `SQLFreeStmt()` is called with the SQL_RESET_PARAMS option, or
- `SQLFreeHandle()` is called with *HandleType* set to SQL_HANDLE_STMT, or `SQLFreeStmt()` is called with the SQL_DROP option, or
- `SQLBindParameter()` is called again for the same *ParameterNumber*, or
- `SQLSetDescField()` is called, with the associated APD descriptor handle, to set SQL_DESC_COUNT in the header field of the APD to zero (0).

A parameter can only be bound to either a file or a storage location, not both. The most recent parameter binding function call determines the bind that is in effect.

Parameter type:

The *InputOutputType* argument specifies the type of the parameter. All parameters in the SQL statements that do not call procedures are input parameters. Parameters in stored procedure calls can be input, input/output, or output parameters. Even though the DB2 stored procedure argument convention typically implies that all procedure arguments are input/output, the application programmer can still choose to specify more exactly the input or output nature on the `SQLBindParameter()` to follow a more rigorous coding style.

- If an application cannot determine the type of a parameter in a procedure call, set *InputOutputType* to SQL_PARAM_INPUT; if the data source returns a value for the parameter, DB2 CLI discards it.
- If an application has marked a parameter as SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT and the data source does not return a value, DB2 CLI sets the *StrLen_or_IndPtr* buffer to SQL_NULL_DATA.
- If an application marks a parameter as SQL_PARAM_OUTPUT, data for the parameter is returned to the application after the CALL statement has been processed. If the *ParameterValuePtr* and *StrLen_or_IndPtr* arguments are both null pointers, DB2 CLI discards the output value. If the data source does not return a value for an output parameter, DB2 CLI sets the *StrLen_or_IndPtr* buffer to SQL_NULL_DATA.
- For this function, *ParameterValuePtr* and *StrLen_or_IndPtr* are deferred arguments. In the case where *InputOutputType* is set to SQL_PARAM_INPUT or SQL_PARAM_INPUT_OUTPUT, the storage locations must be valid and contain input data values when the statement is executed. This means either keeping the `SQLExecDirect()` or `SQLExecute()` call in the same procedure scope as the `SQLBindParameter()` calls, or, these storage locations must be dynamically allocated or statically / globally declared.

  Similarly, if *InputOutputType* is set to SQL_PARAM_OUTPUT or SQL_PARAM_INPUT_OUTPUT, the *ParameterValuePtr* and *StrLen_or_IndPtr* buffer locations must remain valid until the CALL statement has been executed.

ParameterValuePtr and StrLen_or_IndPtr arguments:

ParameterValuePtr and *StrLen_or_IndPtr* are deferred arguments, so the storage locations they point to must be valid and contain input data values when the statement is executed. This means either keeping the `SQLExecDirect()` or `SQLExecute()` call in the same application function scope as the `SQLBindParameter()` calls, or dynamically allocating or statically or globally declaring these storage locations.

Since the data in the variables referenced by *ParameterValuePtr* and *StrLen_or_IndPtr* is not verified until the statement is executed, data content or format errors are not detected or reported until `SQLExecute()` or `SQLExecDirect()` is called.

An application can pass the value for a parameter either in the *ParameterValuePtr* buffer or with one or more calls to `SQLPutData()`. In the latter case, these parameters are data-at-execution parameters. The application informs DB2 CLI of a data-at-execution parameter by placing the SQL_DATA_AT_EXEC value in the buffer pointed to by *StrLen_or_IndPtr*. It sets the *ParameterValuePtr* input argument

to a 32-bit value which will be returned on a subsequent SQLParamData() call and can be used to identify the parameter position.

When SQLBindParameter() is used to bind an application variable to an output parameter for a stored procedure, DB2 CLI can provide some performance enhancement if the *ParameterValuePtr* buffer is placed consecutively in memory after the *StrLen_or_IndPtr* buffer. For example:

```
struct {  SQLINTEGER  StrLen_or_IndPtr;
          SQLCHAR     ParameterValuePtr[MAX_BUFFER];
       } column;
```

BufferLength argument:

For character and binary C data, the *BufferLength* argument specifies the length of the *ParameterValuePtr* buffer if it is a single element; or, if the application calls SQLSetStmtAttr() with the SQL_ATTR_PARAMSET_SIZE attribute to specify multiple values for each parameter, *BufferLength* is the length of *each* element in the *ParameterValuePtr* array, including the null-terminator. If the application specifies multiple values, *BufferLength* is used to determine the location of values in the *ParameterValuePtr* array. For all other types of C data, the *BufferLength* argument is ignored.

ColumnSize argument:

When actual size of the target column or output parameter is not known, the application can specify 0 for the length of the column. (*ColumnSize* set to 0).

If the column's data type is of fixed-length, the DB2 CLI driver will base the length from the data type itself. However, setting *ColumnSize* to 0 means different things when the data type is of type character, binary string or large object:

Input parameter
A 0 *ColumnSize* means that DB2 CLI will use the maximum length for the SQL type provided as the size of the column or stored procedure parameter. DB2 CLI will perform any necessary conversions using this size.

Output parameter (stored procedures only)
A 0 *ColumnSize* means that DB2 CLI will use *BufferLength* as the parameter's size. Note that this means that the stored procedure must not return more than *BufferLength* bytes of data or a truncation error will occur.

For Input-output parameter (store procedures only)
A 0 *ColumnSize* means that DB2 CLI will set both the input and output to *BufferLength* as the target parameter. This means that the input data will be converted to this new size if necessary before being sent to the stored procedure and at most *BufferLength* bytes of data are expected to be returned.

Setting *ColumnSize* to 0 is not recommended unless it is required; it causes DB2 CLI to perform costly checking for the length of the data at run time.

Descriptors:

How a parameter is bound is determined by fields of the APD and IPD. The arguments in SQLBindParameter() are used to set those descriptor fields. The fields can also be set by the SQLSetDescField() functions, although SQLBindParameter() is more efficient to use because the application does not have to obtain a descriptor handle to call SQLBindParameter().

Note: Calling SQLBindParameter() for one statement can affect other statements. This occurs when the APD associated with the statement is explicitly allocated and is also associated with other statements. Because SQLBindParameter() modifies the fields of the APD, the modifications apply to all statements with which this descriptor is associated. If this is not the required behavior, the application should dissociate the descriptor from the other statements before calling SQLBindParameter().

Conceptually, SQLBindParameter() performs the following steps in sequence:
- Calls SQLGetStmtAttr() to obtain the APD handle.
- Calls SQLGetDescField() to get the SQL_DESC_COUNT header field from the APD, and if the value of the *ParameterNumber* argument exceeds the value of SQL_DESC_COUNT, calls SQLSetDescField() to increase the value of SQL_DESC_COUNT to *ParameterNumber*.
- Calls SQLSetDescField() multiple times to assign values to the following fields of the APD:
  - Sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of *ValueType*, except that if *ValueType* is one of the concise identifiers of a datetime, it sets SQL_DESC_TYPE to SQL_DATETIME, sets SQL_DESC_CONCISE_TYPE to the concise identifier, and sets SQL_DESC_DATETIME_INTERVAL_CODE to the corresponding datetime subcode.
  - Sets the SQL_DESC_DATA_PTR field to the value of *ParameterValue*.
  - Sets the SQL_DESC_OCTET_LENGTH_PTR field to the value of *StrLen_or_Ind*.
  - Sets the SQL_DESC_INDICATOR_PTR field also to the value of *StrLen_or_Ind*.

  The *StrLen_or_Ind* parameter specifies both the indicator information and the length for the parameter value.
- Calls SQLGetStmtAttr() to obtain the IPD handle.
- Calls SQLGetDescField() to get the IPD's SQL_DESC_COUNT field, and if the value of the *ParameterNumber* argument exceeds the value of SQL_DESC_COUNT, calls SQLSetDescField() to increase the value of SQL_DESC_COUNT to *ParameterNumber*.
- Calls SQLSetDescField() multiple times to assign values to the following fields of the IPD:
  - Sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of *ParameterType*, except that if *ParameterType* is one of the concise identifiers of a datetime, it sets SQL_DESC_TYPE to SQL_DATETIME, sets SQL_DESC_CONCISE_TYPE to the concise identifier, and sets SQL_DESC_DATETIME_INTERVAL_CODE to the corresponding datetime subcode.
  - Sets one or more of SQL_DESC_LENGTH, SQL_DESC_PRECISION, and SQL_DESC_SCALE as appropriate for *ParameterType*.

If the call to SQLBindParameter() fails, the content of the descriptor fields that it would have set in the APD are undefined, and the SQL_DESC_COUNT field of the APD is unchanged. In addition, the SQL_DESC_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE, and SQL_DESC_TYPE fields of the appropriate record in the IPD are undefined and the SQL_DESC_COUNT field of the IPD is unchanged.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR

- SQL_INVALID_HANDLE

Diagnostics:

Table 11. SQLBindParameter SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The conversion from the data value identified by the *ValueType* argument to the data type identified by the *ParameterType* argument is not a meaningful conversion. (For example, conversion from SQL_C_DATE to SQL_DOUBLE.)
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	The value specified by the argument *ParameterNumber* not a valid data type or SQL_C_DEFAULT.
HY004	SQL data type out of range.	The value specified for the argument *ParameterType* is not a valid SQL data type.
HY009	Invalid argument value.	The argument *ParameterValuePtr* was a null pointer and the argument *StrLen_or_IndPtr* was a null pointer, and *InputOutputType* is not SQL_PARAM_OUTPUT.
HY010	Function sequence error.	Function was called after `SQLExecute()` or `SQLExecDirect()` had returned SQL_NEED_DATA, but data has not been sent for all *data-at-execution* parameters.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check was not consistent.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* was less than 0.
HY093	Invalid parameter number.	The value specified for the argument *ValueType* was less than 1 or greater than the maximum number of parameters supported by the server.
HY094	Invalid scale value.	The value specified for *ParameterType* was either SQL_DECIMAL or SQL_NUMERIC and the value specified for *DecimalDigits* was less than 0 or greater than the value for the argument *ParamDef* (precision). The value specified for *ParameterType* was SQL_C_TIMESTAMP and the value for *ParameterType* was either SQL_CHAR or SQL_VARCHAR and the value for *DecimalDigits* was less than 0 or greater than 6.
HY104	Invalid precision value.	The value specified for *ParameterType* was either SQL_DECIMAL or SQL_NUMERIC and the value specified for *ParamDef* was less than 1.
HY105	Invalid parameter type.	*InputOutputType* is not one of SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT.

Table 11. SQLBindParameter SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HYC00	Driver not capable.	DB2 CLI or data source does not support the conversion specified by the combination of the value specified for the argument *ValueType* and the value specified for the argument *ParameterType*. The value specified for the argument *ParameterType* is not supported by either DB2 CLI or the data source.

Restrictions:

`SQLBindParameter()` replaces the deprecated `SQLSetParam()` API in DB2 CLI V5 and above, and ODBC 2.0 and above.

An additional value for *StrLen_or_IndPtr*, SQL_DEFAULT_PARAM, was introduced in ODBC 2.0, to indicate that the procedure is to use the default value of a parameter, rather than a value sent from the application. Since DB2 stored procedure arguments do not support default values, specification of this value for *StrLen_or_IndPtr* argument will result in an error when the CALL statement is executed since the SQL_DEFAULT_PARAM value will be considered an invalid length.

ODBC 2.0 also introduced the SQL_LEN_DATA_AT_EXEC(*length*) macro to be used with the *StrLen_or_IndPtr* argument. The macro is used to specify the sum total length of the entire data that would be sent for character or binary C data via the subsequent `SQLPutData()` calls. Since the DB2 ODBC driver does not need this information, the macro is not needed. An ODBC application calls `SQLGetInfo()` with the SQL_NEED_LONG_DATA_LEN option to check if the driver needs this information. The DB2 ODBC driver will return 'N' to indicate that this information is not needed by `SQLPutData()`.

Example:

```
SQLSMALLINT parameter1 = 0;

/* ... */

cliRC = SQLBindParameter(hstmt,
                         1,
                         SQL_PARAM_INPUT,
                         SQL_C_SHORT,
                         SQL_SMALLINT,
                         0,
                         0,
                         &parameter1,
                         0,
                         NULL);
```

Related concepts:

- "Parameter marker binding in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Calling stored procedures from CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter" on page 20
- "SQLParamData function (CLI) - Get next parameter for which a data value is needed" on page 239
- "SQLParamOptions function (CLI) - Specify an input array for a parameter" on page 242
- "SQLPutData function (CLI) - Passing data value for a parameter" on page 261
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbread.c -- How to read data from tables"
- "dbuse.c -- How to use a database"

SQLBrowseConnect function (CLI) - Get required attributes to connect to data source

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 1	

`SQLBrowseConnect()` supports an iterative method of discovering and enumerating the attributes and attribute values required to connect to a data source. Each call to `SQLBrowseConnect()` returns successive levels of attributes and attribute values. When all levels have been enumerated, a connection to the data source is completed and a complete connection string is returned by `SQLBrowseConnect()`. A return code of SQL_SUCCESS or SQL_SUCCESS_WITH_INFO indicates that all connection information has been specified and the application is now connected to the data source.

Unicode Equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLBrowseConnectW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN  SQLBrowseConnect (
             SQLHDBC      ConnectionHandle,              /* hdbc */
             SQLCHAR      *InConnectionString,           /* *szConnStrIn */
             SQLSMALLINT  InConnectionStringLength,      /* dbConnStrIn */
             SQLCHAR      *OutConnectionString,          /* *szConnStrOut */
             SQLSMALLINT  OutConnectionStringCapacity,   /* dbConnStrOutMax */
             SQLSMALLINT  *OutConnectionStringLengthPtr); /* *pcbConnStrOut */
```

Function Arguments:

Table 12. SQLBrowseConnect arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLCHAR *	*InConnectionString*	input	Browse request connection string (see "InConnectionString argument" on page 38).

Table 12. SQLBrowseConnect arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*InConnectionStringLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store **InConnectionString*.
SQLCHAR *	*OutConnectionString*	output	Pointer to a buffer in which to return the browse result connection string (see "OutConnectionString argument" on page 38).
SQLSMALLINT	*OutConnectionString Capacity*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the **OutConnectionString* buffer.
SQLSMALLINT *	*OutConnectionString LengthPtr*	output	The total number of elements (excluding the null termination character) available to return in **OutConnectionString*. If the number of elements available to return is greater than or equal to *OutConnectionStringCapacity*, the connection string in **OutConnectionString* is truncated to *OutConnectionStringCapacity* minus the length of a null termination character.

Usage:

`SQLBrowseConnect()` requires an allocated connection. If `SQLBrowseConnect()` returns SQL_ERROR, outstanding connection information is discarded, and the connection is returned to an unconnected state.

When `SQLBrowseConnect()` is called for the first time on a connection, the browse request connection string must contain the DSN keyword.

On each call to `SQLBrowseConnect()`, the application specifies the connection attribute values in the browse request connection string. DB2 CLI returns successive levels of attributes and attribute values in the browse result connection string; it returns SQL_NEED_DATA as long as there are connection attributes that have not yet been enumerated in the browse request connection string. The application uses the contents of the browse result connection string to build the browse request connection string for the next call to `SQLBrowseConnect()`. All mandatory attributes (those not preceded by an asterisk in the *OutConnectionString* argument) must be included in the next call to `SQLBrowseConnect()`. Note that the application cannot simply copy the entire content of previous browse result connection strings when building the current browse request connection string; that is, it cannot specify different values for attributes set in previous levels.

When all levels of connection and their associated attributes have been enumerated, DB2 CLI returns SQL_SUCCESS, the connection to the data source is complete, and a complete connection string is returned to the application. The connection string is suitable to use as an argument for `SQLDriverConnect()` in conjunction with the SQL_DRIVER_NOPROMPT option to establish another connection. The complete connection string cannot be used in another call to `SQLBrowseConnect()`, however; if `SQLBrowseConnect()` were called again, the entire sequence of calls would have to be repeated.

`SQLBrowseConnect()` also returns SQL_NEED_DATA if there are recoverable, nonfatal errors during the browse process, for example, an invalid password supplied by the application or an invalid attribute keyword supplied by the

application. When SQL_NEED_DATA is returned and the browse result connection string is unchanged, an error has occurred and the application can call `SQLGetDiagRec()` to return the SQLSTATE for browse-time errors. This permits the application to correct the attribute and continue the browse.

An application can terminate the browse process at any time by calling `SQLDisconnect()`. DB2 CLI will terminate any outstanding connection information and return the connection to an unconnected state.

InConnectionString argument:

A browse request connection string has the following syntax:

connection-string ::= attribute[] | attribute: connection-string

attribute ::= attribute-keyword=attribute-value
| DRIVER=[{]attribute-value[}]

attribute-keyword ::= DSN | UID | PWD | NEWPWD
| driver-defined-attribute-keyword

attribute-value ::= character-string
driver-defined-attribute-keyword ::= identifier

where
- character-string has zero or more SQLCHAR or SQLWCHAR elements
- identifier has one or more SQLCHAR or SQLWCHAR elements
- attribute-keyword is case insensitive
- attribute-value might be case sensitive
- the value of the **DSN** keyword does not consist solely of blanks
- NEWPWD is used as part of a change password request. The application can either specify the new string to use, for example, NEWPWD=anewpass; or specify NEWPWD=; and rely on a dialog box generated by the DB2 CLI driver to prompt for the new password

Because of connection string and initialization file grammar, keywords and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (\) character. For DB2 CLI Version 2, braces are required around the DRIVER keyword.

If any keywords are repeated in the browse request connection string, DB2 CLI uses the value associated with the first occurrence of the keyword. If the **DSN** and **DRIVER** keywords are included in the same browse request connection string, DB2 CLI uses whichever keyword appears first.

OutConnectionString argument:

The browse result connection string is a list of connection attributes. A connection attribute consists of an attribute keyword and a corresponding attribute value. The browse result connection string has the following syntax:

connection-string ::= attribute[;] | attribute; connection-string

attribute ::= [*]attribute-keyword=attribute-value
attribute-keyword ::= ODBC-attribute-keyword

| driver-defined-attribute-keyword

ODBC-attribute-keyword = {UID | PWD}[:localized-identifier]
driver-defined-attribute-keyword ::= identifier[:localized-identifier]

attribute-value ::= {attribute-value-list} | ?
(The braces are literal; they are returned by DB2 CLI.)
attribute-value-list ::= character-string [:localized-character
string] | character-string [:localized-character string], attribute-value-list

where
- character-string and localized-character string have zero or more SQLCHAR or SQLWCHAR elements
- identifier and localized-identifier have one or more elements; attribute-keyword is case insensitive
- attribute-value might be case sensitive

Because of connection string and initialization file grammar, keywords, localized identifiers, and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (\) character.

The browse result connection string syntax is used according to the following semantic rules:

- If an asterisk (*) precedes an attribute-keyword, the attribute is optional, and can be omitted in the next call to `SQLBrowseConnect()`.
- The attribute keywords **UID** and **PWD** have the same meaning as defined in `SQLDriverConnect()`.
- When connecting to a DB2 database, only **DSN**, **UID** and **PWD** are required. Other keywords can be specified but do not affect the connection.
- ODBC-attribute-keywords and driver-defined-attribute-keywords include a localized or user-friendly version of the keyword. This might be used by applications as a label in a dialog box. However, **UID**, **PWD**, or the identifier alone must be used when passing a browse request string to DB2 CLI.
- The {attribute-value-list} is an enumeration of actual values valid for the corresponding attribute-keyword. Note that the braces ({}) do not indicate a list of choices; they are returned by DB2 CLI. For example, it might be a list of server names or a list of database names.
- If the attribute-value is a single question mark (?), a single value corresponds to the attribute-keyword. For example, UID=JohnS; PWD=Sesame.
- Each call to `SQLBrowseConnect()` returns only the information required to satisfy the next level of the connection process. DB2 CLI associates state information with the connection handle so that the context can always be determined on each call.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 13. SQLBrowseConnect SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The buffer **OutConnectionString* was not large enough to return entire browse result connection string, so the string was truncated. The buffer **OutConnectionStringLengthPtr* contains the length of the untruncated browse result connection string. (Function returns SQL_SUCCESS_WITH_INFO.)
01S00	Invalid connection string attribute.	An invalid attribute keyword was specified in the browse request connection string (*InConnectionString*). (Function returns SQL_NEED_DATA.) An attribute keyword was specified in the browse request connection string (*InConnectionString*) that does not apply to the current connection level. (Function returns SQL_NEED_DATA.)
01S02	Option value changed.	DB2 CLI did not support the specified value of the *ValuePtr* argument in `SQLSetConnectAttr()` and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08001	Unable to connect to data source.	DB2 CLI was unable to establish a connection with the data source.
08002	Connection in use.	The specified connection had already been used to establish a connection with a data source and the connection was open.
08004	The application server rejected establishment of the connection.	The data source rejected the establishment of the connection for implementation defined reasons.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was trying to connect failed before the function completed processing.
28000	Invalid authorization specification.	Either the user identifier or the authorization string or both as specified in the browse request connection string (*InConnectionString*) violated restrictions defined by the data source.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument *InConnectionStringLength* was less than 0 and was not equal to SQL_NTS. The value specified for argument *OutConnectionStringCapacity* was less than 0.

Restrictions:

None.

Example:

```
SQLCHAR connInStr[255]; /* browse request connection string */
SQLCHAR outStr[1025]; /* browse result connection string*/

/* ... */

cliRC = SQL_NEED_DATA;
while (cliRC == SQL_NEED_DATA)
{
  /* get required attributes to connect to data source */
  cliRC = SQLBrowseConnect(hdbc,
                           connInStr,
                           SQL_NTS,
                           outStr,
                           sizeof(outStr),
                           &indicator);
  DBC_HANDLE_CHECK(hdbc, cliRC);

  printf("  So far, have connected %d times to database %s\n",
         count++, db1Alias);
  printf("  Resulting connection string: %s\n", outStr);

  /* if inadequate connection information was provided, exit
     the program */
  if (cliRC == SQL_NEED_DATA)
  {
    printf("  You can provide other connection information "
           "here by setting connInStr\n");
    break;
  }

  /* if the connection was successful, output confirmation */
  if (cliRC == SQL_SUCCESS)
  {
    printf("  Connected to the database %s.\n", db1Alias);
  }
}
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLConnect function (CLI) - Connect to a data source" on page 73
- "SQLDisconnect function (CLI) - Disconnect from a data source" on page 89
- "SQLDriverConnect function (CLI) - (Expanded) Connect to a data source" on page 91
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbcongui.c -- How to connect to a database with a graphical user interface (GUI)"

SQLBuildDataLink function (CLI) - Build DATALINK value

Purpose:

Specification:	DB2 CLI 5.2		ISO CLI

SQLBuildDataLink

`SQLBuildDataLink()` returns a DATALINK value built from input arguments.

Syntax:

```
SQLRETURN   SQLBuildDataLink (
               SQLHSTMT        StatementHandle,     /* hStmt */
               SQLCHAR         *LinkType,           /* *pszLinkType */
               SQLINTEGER      LinkTypeLength,      /* cbLinkType */
               SQLCHAR         *DataLocation,       /* *pszDataLocation */
               SQLINTEGER      DataLocationLength,  /* cbDataLocation */
               SQLCHAR         *Comment,            /* *pszComment */
               SQLINTEGER      CommentLength,       /* cbComment */
               SQLCHAR         *DataLinkValue,      /* *pDataLink */
               SQLINTEGER      BufferLength,        /* cbDataLinkMax */
               SQLINTEGER      *StringLengthPtr);   /* *pcbDataLink */
```

Function arguments:

Table 14. SQLBuildDataLink arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Used only for diagnostic reporting.
SQLCHAR *	*LinkType*	input	Always set to SQL_DATALINK_URL.
SQLINTEGER	*LinkTypeLength*	input	The length of the *LinkType* value.
SQLCHAR *	*DataLocation*	input	The complete URL value to be assigned.
SQLINTEGER	*DataLocationLength*	input	The length of the *DataLocation* value.
SQLCHAR *	*Comment*	input	The comment, if any, to be assigned.
SQLINTEGER	*CommentLength*	input	The length of the *Comment* value.
SQLCHAR *	*DataLinkValue*	output	The DATALINK value that is created by the function.
SQLINTEGER	*BufferLength*	input	Length of the *DataLinkValue* buffer.
SQLINTEGER *	*StringLengthPtr*	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **DataLinkValue*. If *DataLinkValue* is a null pointer, no length is returned. If the number of bytes available to return is greater than BufferLength minus the length of the null-termination character, then SQLSTATE 01004 is returned. In this case, subsequent use of the DATALINK value might fail.

Usage:

The function is used to build a DATALINK value. The maximum length of the string, including the null termination character, will be *BufferLength* bytes.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 15. SQLBuildDataLink() SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
01004	Data truncated.	The data returned in **DataLinkValue* was truncated to be *BufferLength* minus the length of the null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY090	Invalid string or buffer length.	The value specified one of the arguments (*LinkTypeLength*, *DataLocationLength*, or *CommentLength*) was less than 0 but not equal to SQL_NTS or *BufferLength* is less than 0.

Restrictions:

DB2 Data Links Manager is no longer supported for DB2 on Linux®, UNIX® and Windows. Check your server for support.

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLGetDataLinkAttr function (CLI) - Get DataLink attribute value" on page 158

SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows

Purpose:

Specification:	DB2 CLI 6.0	ODBC 3.0	

`SQLBulkOperations()` is used to perform the following operations on a keyset-driven cursor:

- Add new rows
- Update a set of rows where each row is identified by a bookmark
- Delete a set of rows where each row is identified by a bookmark
- Fetch a set of rows where each row is identified by a bookmark

Syntax:

```
SQLRETURN   SQLBulkOperations    (
                SQLHSTMT          StatementHandle,
                SQLSMALLINT       Operation);
```

Function arguments:

Table 16. SQLBulkOperations arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLSMALLINT	*Operation*	Input	Operation to perform: • SQL_ADD • SQL_UPDATE_BY_BOOKMARK • SQL_DELETE_BY_BOOKMARK • SQL_FETCH_BY_BOOKMARK

Usage:

An application uses `SQLBulkOperations()` to perform the following operations on the base table or view that corresponds to the current query in a keyset-driven cursor:

- Add new rows
- Update a set of rows where each row is identified by a bookmark
- Delete a set of rows where each row is identified by a bookmark
- Fetch a set of rows where each row is identified by a bookmark

A generic application should first ensure that the required bulk operation is supported. To do so, it can call `SQLGetInfo()` with an *InfoType* of SQL_DYNAMIC_CURSOR_ATTRIBUTES1 and SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (to see if SQL_CA1_BULK_UPDATE_BY_BOOKMARK is returned, for instance).

After a call to `SQLBulkOperations()`, the block cursor position is undefined. The application has to call `SQLFetchScroll()` to set the cursor position. An application should only call `SQLFetchScroll()` with a *FetchOrientation* argument of SQL_FETCH_FIRST, SQL_FETCH_LAST, SQL_FETCH_ABSOLUTE, or SQL_FETCH_BOOKMARK. The cursor position is undefined if the application calls `SQLFetch()`, or `SQLFetchScroll()` with a *FetchOrientation* argument of SQL_FETCH_PRIOR, SQL_FETCH_NEXT, or SQL_FETCH_RELATIVE.

A column can be ignored in bulk operations (calls to `SQLBulkOperations()`). To do so, call `SQLBindCol()` and set the column length/indicator buffer (*StrLen_or_IndPtr*) to SQL_COLUMN_IGNORE. This does not apply to SQL_DELETE_BY_BOOKMARK bulk operation.

It is not necessary for the application to set the SQL_ATTR_ROW_OPERATION_PTR statement attribute when calling `SQLBulkOperations()` because rows cannot be ignored when performing bulk operations with this function.

The buffer pointed to by the SQL_ATTR_ROWS_FETCHED_PTR statement attribute contains the number of rows affected by a call to `SQLBulkOperations()`.

When the *Operation* argument is SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the select-list of the query specification associated with the cursor contains more than one reference to the same column, an error is generated.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_STILL_EXECUTING

- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 17. SQLBulkOperations SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The *Operation* argument was SQL_FETCH_BY_BOOKMARK, and string or binary data returned for a column or columns with a data type of SQL_C_CHAR or SQL_C_BINARY resulted in the truncation of non-blank character or non-NULL binary data.
01S07	Invalid conversion.	The *Operation* argument was SQL_FETCH_BY_BOOKMARK, the data type of the application buffer was not SQL_C_CHAR or SQL_C_BINARY, and the data returned to application buffers for one or more columns was truncated. (For numeric C data types, the fractional part of the number was truncated. For time and timestamp data types, the fractional portion of the time was truncated.) (Function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation.	The *Operation* argument was SQL_FETCH_BY_BOOKMARK, and the data value of a column in the result set could not be converted to the data type specified by the *TargetType* argument in the call to `SQLBindCol()`. The *Operation* argument was SQL_UPDATE_BY_BOOKMARK or SQL_ADD, and the data value in the application buffers could not be converted to the data type of a column in the result set.
07009	Invalid descriptor index.	The argument *Operation* was SQL_ADD and a column was bound with a column number greater than the number of columns in the result set, or the column number was less than 0.
21S02	Degree of derived table does not match column list.	The argument *Operation* was SQL_UPDATE_BY_BOOKMARK; and no columns were updatable because all columns were either unbound, read-only, or the value in the bound length/indicator buffer was SQL_COLUMN_IGNORE.
22001	String data right truncation.	The assignment of a character or binary value to a column in the result set resulted in the truncation of non-blank (for characters) or non-null (for binary) characters or bytes.
22003	Numeric value out of range.	The *Operation* argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the assignment of a numeric value to a column in the result set caused the whole (as opposed to fractional) part of the number to be truncated. The argument *Operation* was SQL_FETCH_BY_BOOKMARK, and returning the numeric value for one or more bound columns would have caused a loss of significant digits.
22007	Invalid datetime format.	The *Operation* argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the assignment of a date or timestamp value to a column in the result set caused the year, month, or day field to be out of range. The argument *Operation* was SQL_FETCH_BY_BOOKMARK, and returning the date or timestamp value for one or more bound columns would have caused the year, month, or day field to be out of range.

Table 17. SQLBulkOperations SQLSTATEs (continued)

SQLSTATE	Description	Explanation
22008	Date/time field overflow.	The *Operation* argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the performance of datetime arithmetic on data being sent to a column in the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar. The *Operation* argument was SQL_FETCH_BY_BOOKMARK, and the performance of datetime arithmetic on data being retrieved from the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar.
22018	Invalid character value for cast specification.	The *Operation* argument was SQL_FETCH_BY_BOOKMARK; the C type was an exact or approximate numeric or datetime data type; the SQL type of the column was a character data type; and the value in the column was not a valid literal of the bound C type. The argument *Operation* was SQL_ADD or SQL_UPDATE_BY_BOOKMARK; the SQL type was an exact or approximate numeric or datetime data type; the C type was SQL_C_CHAR; and the value in the column was not a valid literal of the bound SQL type.
23000	Integrity constraint violation.	The *Operation* argument was SQL_ADD, SQL_DELETE_BY_BOOKMARK, or SQL_UPDATE_BY_BOOKMARK, and an integrity constraint was violated. The *Operation* argument was SQL_ADD, and a column that was not bound is defined as NOT NULL and has no default. The *Operation* argument was SQL_ADD, the length specified in the bound *StrLen_or_IndPtr* buffer was SQL_COLUMN_IGNORE, and the column did not have a default value.
24000	Invalid cursor state.	The *StatementHandle* was in an executed state but no result set was associated with the *StatementHandle*. `SQLFetch()` or `SQLFetchScroll()` was not called by the application after `SQLExecute()` or `SQLExecDirect()`.
40001	Serialization failure.	The transaction was rolled back due to a resource deadlock with another transaction.
40003	Statement completion unknown.	The associated connection failed during the execution of this function and the state of the transaction cannot be determined.
42000	Syntax error or access violation.	DB2 CLI was unable to lock the row as needed to perform the operation requested in the *Operation* argument.
44000	WITH CHECK OPTION violation.	The *Operation* argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, and the insert or update was performed on a viewed table or a table derived from the viewed table which was created by specifying WITH CHECK OPTION, such that one or more rows affected by the insert or update will no longer be present in the viewed table.

Table 17. SQLBulkOperations SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation error.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called For the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY011	Operation invalid at this time.	The SQL_ATTR_ROW_STATUS_PTR statement attribute was set between calls to `SQLFetch()` or `SQLFetchScroll()` and `SQLBulkOperations`.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of this function.

Table 17. SQLBulkOperations SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The *Operation* argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. The *Operation* argument was SQL_ADD or SQL_UPDATE_BY_BOOKMARK, a data value was not a null pointer; the C data type was SQL_C_BINARY or SQL_C_CHAR; and the column length value was less than 0, but not equal to SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. The value in a length/indicator buffer was SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a long data type; and the SQL_NEED_LONG_DATA_LEN information type in `SQLGetInfo()` was "Y". The *Operation* argument was SQL_ADD, the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_VARIABLE, and column 0 was bound to a buffer whose length was not equal to the maximum length for the bookmark for this result set. (This length is available in the SQL_DESC_OCTET_LENGTH field of the IRD, and can be obtained by calling `SQLDescribeCol()`, `SQLColAttribute()`, or `SQLGetDescField()`.)
HY092	Invalid attribute identifier.	The value specified for the *Operation* argument was invalid. The *Operation* argument was SQL_ADD, SQL_UPDATE_BY_BOOKMARK, or SQL_DELETE_BY_BOOKMARK, and the SQL_ATTR_CONCURRENCY statement attribute was set to SQL_CONCUR_READ_ONLY. The *Operation* argument was SQL_DELETE_BY_BOOKMARK, SQL_FETCH_BY_BOOKMARK, or SQL_UPDATE_BY_BOOKMARK, and the bookmark column was not bound or the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
HYC00	Optional feature not implemented.	DB2 CLI or data source does not support the operation requested in the *Operation* argument.
HYT00	Timeout expired.	The query timeout period expired before the data source returned the result set. The timeout period is set through `SQLSetStmtAttr()` with an *Attribute* argument of SQL_ATTR_QUERY_TIMEOUT.
HYT01	Connection timeout expired.	The connection timeout period expired before the data source responded to the request. The connection timeout period is set through `SQLSetConnectAttr()`, SQL_ATTR_CONNECTION_TIMEOUT.

Restrictions:

None.

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Long data for bulk inserts and updates in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Deleting bulk data with bookmarks using SQLBulkOperations() in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Inserting bulk data with bookmarks using SQLBulkOperations() in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Retrieving bulk data with bookmarks using SQLBulkOperations() in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Updating bulk data with bookmarks using SQLBulkOperations() in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "SQLGetInfo function (CLI) - Get general information" on page 180
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLCancel function (CLI) - Cancel statement

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLCancel()` can be used to prematurely terminate the data-at-execution sequence for sending and retrieving long data in pieces.

`SQLCancel()` can also be used to cancel a function called in a different thread.

Syntax:

```
SQLRETURN   SQLCancel        (SQLHSTMT          StatementHandle);  /* hstmt */
```

Function arguments:

Table 18. SQLCancel arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle

Usage:

After `SQLExecDirect()` or `SQLExecute()` returns SQL_NEED_DATA to solicit for values for data-at-execution parameters, `SQLCancel()` can be used to cancel the data-at-execution sequence for sending and retrieving long data in pieces. `SQLCancel()` can be called any time before the final `SQLParamData()` in the

sequence. After the cancellation of this sequence, the application can call SQLExecute() or SQLExecDirect() to re-initiate the data-at-execution sequence.

If no processing is being done on the statement, SQLCancel() has no effect. Applications should not call SQLCancel() to close a cursor, but rather SQLFreeStmt() should be used.

Canceling queries on host databases:

To call SQLCancel() against a server which does not have native interrupt support (such as DB2 Universal Database™ for z/OS® and OS/390®, Version 7 and earlier, and DB2 for iSeries™), the INTERRUPT_ENABLED option must be set when cataloging the DCS database entry for the server.

When the INTERRUPT_ENABLED option is set and SQLCancel() is received by the server, the server drops the connection and rolls back the unit of work. The application receives an SQL30081N error indicating that the connection to the server has been terminated. In order for the application to process additional database requests, the application must establish a new connection with the database server.

Canceling asynchronous processing:

After an application calls a function asynchronously, it calls the function repeatedly to determine whether it has finished processing. If the function is still processing, it returns SQL_STILL_EXECUTING.

After any call to the function that returns SQL_STILL_EXECUTING, an application can call SQLCancel() to cancel the function. If the cancel request is successful, SQL_SUCCESS is returned. This message does not indicate that the function was actually canceled; it indicates that the cancel request was processed. The application must then continue to call the original function until the return code is not SQL_STILL_EXECUTING. If the function was successfully canceled, the return code is for that function is SQL_ERROR and SQLSTATE HY008 (Operation was cancelled.). If the function succeeded by completing its normal processing, the return code is SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. If the function failed for reasons other than cancellation, the return code is SQL_ERROR and an SQLSTATE other than HY008 (Operation was cancelled.).

Canceling functions in multithread applications:

In a multithread application, the application can cancel a function that is running synchronously on a statement. To cancel the function, the application calls SQLCancel() with the same statement handle as that used by the target function, but on a different thread. How the function is canceled depends upon the operating system. The return code of the SQLCancel() call indicates only whether DB2 CLI processed the request successfully. Only SQL_SUCCESS or SQL_ERROR can be returned; no SQLSTATEs are returned. If the original function is canceled, it returns SQL_ERROR and SQLSTATE HY008 (Operation was cancelled.).

If an SQL statement is being executed when SQLCancel() is called on another thread to cancel the statement execution, it is possible that the execution succeeds and returns SQL_SUCCESS, while the cancel is also successful. In this case, DB2 CLI assumes that the cursor opened by the statement execution is closed by the cancel, so the application will not be able to use the cursor.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_INVALID_HANDLE
- SQL_ERROR

Note: SQL_SUCCESS means that the cancel request was processed, not that the function call was canceled.

Diagnostics:

Table 19. SQLCancel SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY018	Server declined cancel request.	The server declined the cancel request.
HY506	Error closing a file.	An error occurred when closing the temporary file generated by DB2 CLI when inserting LOB data in pieces using `SQLParamData()/SQLPutData()`.

Restrictions:

None.

Example:

```
/* cancel the SQL_DATA_AT_EXEC state for hstmt */
cliRC = SQLCancel(hstmt);
```

Related concepts:
- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Large object usage in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Long data for bulk inserts and updates in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Multithreaded CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "DCS directory values" in *DB2 Connect User's Guide*

Related tasks:
- "Specifying parameter values at execute time for long data manipulation in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101

- "SQLExecute function (CLI) - Execute a statement" on page 106
- "SQLParamData function (CLI) - Get next parameter for which a data value is needed" on page 239
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:
- "dtlob.c -- How to read and write LOB data"

SQLCloseCursor function (CLI) - Close cursor and discard pending results

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLCloseCursor()` closes a cursor that has been opened on a statement and discards pending results.

Syntax:

```
SQLRETURN   SQLCloseCursor (SQLHSTMT     StatementHandle);  /* hStmt */
```

Function arguments:

Table 20. SQLCloseCursor arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle

Usage:

After an application calls `SQLCloseCursor()`, the application can reopen the cursor later by executing a SELECT statement again with the same or different parameter values. `SQLCloseCursor()` can be called before a transaction is completed.

`SQLCloseCursor()` returns SQLSTATE 24000 (Invalid cursor state) if no cursor is open. Calling `SQLCloseCursor()` is equivalent to calling `SQLFreeStmt()` with the SQL_CLOSE option, with the exception that `SQLFreeStmt()` with SQL_CLOSE has no effect on the application if no cursor is open on the statement, while `SQLCloseCursor()` returns SQLSTATE 24000 (Invalid cursor state).

The statement attribute SQL_ATTR_CLOSE_BEHAVIOR can be used to indicate whether or not DB2 CLI should attempt to release read locks acquired during a cursor's operation when the cursor is closed. If SQL_ATTR_CLOSE_BEHAVIOR is set to SQL_CC_RELEASE then the database manager will attempt to release all read locks (if any) that have been held for the cursor.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 21. SQLCloseCursor SQLSTATEs

SQLSTATE	Description	Explanation
01000	General warning	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state.	No cursor was open on the *StatementHandle*. (This is returned only by DB2 CLI Version 5 or later.)
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	An asynchronously executing function was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()` or `SQLExecDirect()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Example:

```
/* close the cursor */
  cliRC = SQLCloseCursor(hstmt);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "CLOSE statement" in *SQL Reference, Volume 2*
- "SQLCancel function (CLI) - Cancel statement" on page 49
- "SQLFreeHandle function (CLI) - Free handle resources" on page 140
- "SQLMoreResults function (CLI) - Determine if there are more result sets" on page 229
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "Statement attributes (CLI) list" on page 348

Related samples:

- "dtlob.c -- How to read and write LOB data"
- "udfcli.c -- How to work with different types of user-defined functions (UDFs)"

SQLColAttribute function (CLI) - Return a column attribute

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLColAttribute()` returns descriptor information for a column in a result set. Descriptor information is returned as a character string, a 32-bit descriptor-dependent value, or an integer value.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLColAttributeW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

In a Windows 64-bit environment, the syntax is as follows:

```
SQLRETURN   SQLColAttribute  (
                SQLHSTMT          StatementHandle,       /* hstmt */
                SQLSMALLINT       ColumnNumber,          /* icol */
                SQLSMALLINT       FieldIdentifier,       /* fDescType */
                SQLPOINTER        CharacterAttributePtr, /* rgbDesc */
                SQLSMALLINT       BufferLength,          /* cbDescMax */
                SQLSMALLINT       *StringLengthPtr,      /* pcbDesc */
                SQLLEN            *NumericAttributePtr); /* pfDesc */
```

The syntax for all other platforms is as follows:

```
SQLRETURN   SQLColAttribute  (
                SQLHSTMT          StatementHandle,       /* hstmt */
                SQLSMALLINT       ColumnNumber,          /* icol */
                SQLSMALLINT       FieldIdentifier,       /* fDescType */
                SQLPOINTER        CharacterAttributePtr, /* rgbDesc */
                SQLSMALLINT       BufferLength,          /* cbDescMax */
                SQLSMALLINT       *StringLengthPtr,      /* pcbDesc */
                SQLPOINTER        NumericAttributePtr);  /* pfDesc */
```

Function arguments:

Table 22. SQLColAttribute arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*ColumnNumber*	input	The number of the record in the IRD from which the field value is to be retrieved. This argument corresponds to the column number of result data, ordered sequentially from left to right, starting at 1. Columns can be described in any order. Column 0 can be specified in this argument, but all values except SQL_DESC_TYPE and SQL_DESC_OCTET_LENGTH will return undefined values.
SQLSMALLINT	*FieldIdentifier*	input	The field in row *ColumnNumber* of the IRD that is to be returned (see Table 23 on page 56).

Table 22. SQLColAttribute arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	*CharacterAttributePtr*	output	Pointer to a buffer in which to return the value in the *FieldIdentifier* field of the *ColumnNumber* row of the IRD, if the field is a character string. Otherwise, the field is unused.
SQLINTEGER	*BufferLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the **CharacterAttributePtr* buffer, if the field is a character string. Otherwise, the field is ignored.
SQLSMALLINT *	*StringLengthPtr*	output	Pointer to a buffer in which to return the total number of bytes (excluding the byte count of the null termination character for character data) available to return in **CharacterAttributePtr*. For character data, if the number of bytes available to return is greater than or equal to *BufferLength*, the descriptor information in **CharacterAttributePtr* is truncated to *BufferLength* minus the length of a null termination character and is null-terminated by DB2 CLI. For all other types of data, the value of *BufferLength* is ignored and DB2 CLI assumes the size of **CharacterAttributePtr* is 32 bits.
SQLLEN* (Window 64-bit) or SQLPOINTER	*NumericAttributePtr*	output	Pointer to a buffer in which to return the value in the *FieldIdentifier* field of the *ColumnNumber* row of the IRD, if the field is a numeric descriptor type, such as SQL_DESC_COLUMN_LENGTH. Otherwise, the field is unused.

Usage:

`SQLColAttribute()` returns information either in **NumericAttributePtr* or in **CharacterAttributePtr*. Integer information is returned in **NumericAttributePtr* as a 32-bit, signed value; all other formats of information are returned in **CharacterAttributePtr*. When information is returned in **NumericAttributePtr*, DB2 CLI ignores *CharacterAttributePtr*, *BufferLength*, and *StringLengthPtr* When information is returned in **CharacterAttributePtr*, DB2 CLI ignores *NumericAttributePtr*.

`SQLColAttribute()` returns values from the descriptor fields of the IRD. The function is called with a statement handle rather than a descriptor handle. The values returned by `SQLColAttribute()` for the *FieldIdentifier* values listed below can also be retrieved by calling `SQLGetDescField()` with the appropriate IRD handle.

The currently defined descriptor types, the version of DB2 CLI in which they were introduced (perhaps with a different name), and the arguments in which information is returned for them are shown below; it is expected that more descriptor types will be defined to take advantage of different data sources.

DB2 CLI must return a value for each of the descriptor types. If a descriptor type does not apply to a data source, then, unless otherwise stated, DB2 CLI returns 0 in **StringLengthPtr* or an empty string in **CharacterAttributePtr*.

The following table lists the descriptor types returned by SQLColAttribute().

Table 23. SQLColAttribute arguments

FieldIdentifier	**Information returned in**	**Description**
SQL_DESC_AUTO_UNIQUE_VALUE (DB2 CLI v2)	Numeric AttributePtr	Indicates if the column data type is an auto increment data type. SQL_FALSE is returned in *NumericAttributePtr* for all DB2 SQL data types. Currently DB2 CLI is not able to determine if a column is an identity column, therefore SQL_FALSE is always returned. This limitation does not fully conform to the ODBC specifications. Future versions of DB2 CLI for Unix and Windows servers will provide auto-unique support.
SQL_DESC_BASE_COLUMN_NAME (DB2 CLI v5)	Character AttributePtr	The base column name for the set column. If a base column name does not exist (as in the case of columns that are expressions), then this variable contains an empty string. This information is returned from the SQL_DESC_BASE_COLUMN_NAME record field of the IRD, which is a read-only field.
SQL_DESC_BASE_TABLE_NAME (DB2 CLI v5)	Character AttributePtr	The name of the base table that contains the column. If the base table name cannot be defined or is not applicable, then this variable contains an empty string.
SQL_DESC_CASE_SENSITIVE (DB2 CLI v2)	Numeric AttributePtr	Indicates if the column data type is a case sensitive data type. Either SQL_TRUE or SQL_FALSE will be returned in *NumericAttributePtr* depending on the data type. Case sensitivity does not apply to graphic data types, SQL_FALSE is returned. SQL_FALSE is returned for non-character data types and for the XML data type.
SQL_DESC_CATALOG_NAME (DB2 CLI v2)	Character AttributePtr	An empty string is returned since DB2 CLI only supports two part naming for a table.
SQL_DESC_CONCISE_TYPE (DB2 CLI v5)	Numeric AttributePtr	The concise data type. For the datetime data types, this field returns the concise data type, for example, SQL_TYPE_TIME. This information is returned from the SQL_DESC_CONCISE_TYPE record field of the IRD.
SQL_DESC_COUNT (DB2 CLI v2)	Numeric AttributePtr	The number of columns in the result set is returned in *NumericAttributePtr*.
SQL_DESC_DISPLAY_SIZE (DB2 CLI v2)	Numeric AttributePtr	The maximum number of bytes needed to display the data in character form is returned in *NumericAttributePtr*. Refer to the data type display size table for the display size of each of the column types.

Table 23. SQLColAttribute arguments (continued)

FieldIdentifier	**Information returned in**	**Description**
SQL_DESC_DISTINCT_TYPE (DB2 CLI v2)	Character AttributePtr	The user defined distinct type name of the column is returned in *CharacterAttributePtr*. If the column is a built-in SQL type and not a user defined distinct type, an empty string is returned. **Note:** This is an IBM defined extension to the list of descriptor attributes defined by ODBC.
SQL_DESC_FIXED_PREC_SCALE (DB2 CLI v2)	Numeric AttributePtr	SQL_TRUE if the column has a fixed precision and non-zero scale that are data-source-specific. SQL_FALSE if the column does not have a fixed precision and non-zero scale that are data-source-specific. SQL_FALSE is returned in *NumericAttributePtr* for all DB2 SQL data types.
SQL_DESC_LABEL (DB2 CLI v2)	Character AttributePtr	The column label is returned in *CharacterAttributePtr*. If the column does not have a label, the column name or the column expression is returned. If the column is unlabeled and unnamed, an empty string is returned.
SQL_DESC_LENGTH (DB2 CLI v2)	Numeric AttributePtr	A numeric value that is either the maximum or actual element (SQLCHAR or SQLWCHAR) length of a character string or binary data type. It is the maximum element length for a fixed-length data type, or the actual element length for a variable-length data type. Its value always excludes the null termination byte that ends the character string. This information is returned from the SQL_DESC_LENGTH record field of the IRD. This value is 0 for the XML data type.
SQL_DESC_LITERAL_PREFIX (DB2 CLI v5)	Character AttributePtr	This VARCHAR(128) record field contains the character or characters that DB2 CLI recognizes as a prefix for a literal of this data type. This field contains an empty string for a data type for which a literal prefix is not applicable.
SQL_DESC_LITERAL_SUFFIX (DB2 CLI v5)	Character AttributePtr	This VARCHAR(128) record field contains the character or characters that DB2 CLI recognizes as a suffix for a literal of this data type. This field contains an empty string for a data type for which a literal suffix is not applicable.
SQL_DESC_LOCAL_TYPE_NAME (DB2 CLI v5)	Character AttributePtr	This VARCHAR(128) record field contains any localized (native language) name for the data type that might be different from the regular name of the data type. If there is no localized name, then an empty string is returned. This field is for display purposes only. The character set of the string is locale-dependent and is typically the default character set of the server.

Table 23. SQLColAttribute arguments (continued)

FieldIdentifier	**Information returned in**	**Description**
SQL_DESC_NAME (DB2 CLI v2)	Character AttributePtr	The name of the column *ColumnNumber* is returned in *CharacterAttributePtr*. If the column is an expression, then the column number is returned. In either case, SQL_DESC_UNNAMED is set to SQL_NAMED. If there is no column name or a column alias, an empty string is returned and SQL_DESC_UNNAMED is set to SQL_UNNAMED. This information is returned from the SQL_DESC_NAME record field of the IRD.
SQL_DESC_NULLABLE (DB2 CLI v2)	Numeric AttributePtr	If the column identified by *ColumnNumber* can contain nulls, then SQL_NULLABLE is returned in *NumericAttributePtr*. If the column is constrained not to accept nulls, then SQL_NO_NULLS is returned in *NumericAttributePtr*. This information is returned from the SQL_DESC_NULLABLE record field of the IRD.
SQL_DESC_NUM_PREX_RADIX (DB2 CLI v5)	Numeric AttributePtr	• If the data type in the SQL_DESC_TYPE field is an approximate data type, this SQLINTEGER field contains a value of 2 because the SQL_DESC_PRECISION field contains the number of bits. • If the data type in the SQL_DESC_TYPE field is an exact numeric data type, this field contains a value of 10 because the SQL_DESC_PRECISION field contains the number of decimal digits.
SQL_DESC_OCTET_LENGTH (DB2 CLI v2)	Numeric AttributePtr	The number of *bytes* of data associated with the column is returned in *NumericAttributePtr*. This is the length in bytes of data transferred on the fetch or `SQLGetData()` for this column if SQL_C_DEFAULT is specified as the C data type. Refer to data type length table for the length of each of the SQL data types. If the column identified in *ColumnNumber* is a fixed length character or binary string, (for example, SQL_CHAR or SQL_BINARY) the actual length is returned. If the column identified in *ColumnNumber* is a variable length character or binary string, (for example, SQL_VARCHAR or SQL_BLOB) the maximum length is returned. If the column identified in *ColumnNumber* is of type SQL_XML, 0 is returned.

Table 23. SQLColAttribute arguments (continued)

FieldIdentifier	**Information returned in**	**Description**
SQL_DESC_PRECISION (DB2 CLI v2)	Numeric AttributePtr	The precision in units of digits is returned in *NumericAttributePtr* if the column is SQL_DECIMAL, SQL_NUMERIC, SQL_DOUBLE, SQL_FLOAT, SQL_INTEGER, SQL_REAL or SQL_SMALLINT. If the column is a character SQL data type, then the precision returned in *NumericAttributePtr*, indicates the maximum number of SQLCHAR or SQLWCHAR elements the column can hold. If the column is a graphic SQL data type, then the precision returned in *NumericAttributePtr*, indicates the maximum number of double-byte elements the column can hold. If the column is the XML data type, the precision is 0. Refer to data type precision table for the precision of each of the SQL data types. This information is returned from the SQL_DESC_PRECISION record field of the IRD.
SQL_DESC_SCALE (DB2 CLI v2)	Numeric AttributePtr	The scale attribute of the column is returned. Refer to the data type scale table for the scale of each of the SQL data types. This information is returned from the SCALE record field of the IRD.
SQL_DESC_SCHEMA_NAME (DB2 CLI v2)	Character AttributePtr	The schema of the table that contains the column is returned in *CharacterAttributePtr*. An empty string is returned as DB2 CLI is unable to determine this attribute.
SQL_DESC_SEARCHABLE (DB2 CLI v2)	Numeric AttributePtr	Indicates if the column data type is searchable: • SQL_PRED_NONE (SQL_UNSEARCHABLE in DB2 CLI v2) if the column cannot be used in a WHERE clause. • SQL_PRED_CHAR (SQL_LIKE_ONLY in DB2 CLI v2) if the column can be used in a WHERE clause only with the LIKE predicate. • SQL_PRED_BASIC (SQL_ALL_EXCEPT_LIKE in DB2 CLI v2) if the column can be used in a WHERE clause with all comparison operators except LIKE. • SQL_SEARCHABLE if the column can be used in a WHERE clause with any comparison operator.
SQL_DESC_TABLE_NAME (DB2 CLI v2)	Character AttributePtr	An empty string is returned as DB2 CLI cannot determine this attribute.

Table 23. SQLColAttribute arguments (continued)

FieldIdentifier	**Information returned in**	**Description**
SQL_DESC_TYPE (DB2 CLI v2)	Numeric AttributePtr	The SQL data type of the column identified in *ColumnNumber* is returned in *NumericAttributePtr*. The possible values returned are listed in table of symbolic and default data types for CLI. When *ColumnNumber* is equal to 0, SQL_BINARY is returned for variable-length bookmarks, and SQL_INTEGER is returned for fixed-length bookmarks. For the datetime data types, this field returns the verbose data type, for example, SQL_DATETIME. This information is returned from the SQL_DESC_TYPE record field of the IRD.
SQL_DESC_TYPE_NAME (DB2 CLI v2)	Character AttributePtr	The type of the column (as entered in an SQL statement) is returned in *CharacterAttributePtr*. For information on each data type refer to the list of symbolic and default data types for CLI.
SQL_DESC_UNNAMED (DB2 CLI v5)	Numeric AttributePtr	SQL_NAMED or SQL_UNNAMED. If the SQL_DESC_NAME field of the IRD contains a column alias, or a column name, SQL_NAMED is returned. If there is no column name or a column alias, SQL_UNNAMED is returned. This information is returned from the SQL_DESC_UNNAMED record field of the IRD.
SQL_DESC_UNSIGNED (DB2 CLI v2)	Numeric AttributePtr	Indicates if the column data type is an unsigned type or not. SQL_TRUE is returned in *NumericAttributePtr* for all non-numeric data types, SQL_FALSE is returned for all numeric data types.
SQL_DESC_UPDATABLE (DB2 CLI v2)	Numeric AttributePtr	Indicates if the column data type is an updateable data type: • SQL_ATTR_READWRITE_UNKNOWN is returned in *NumericAttributePtr* for all DB2 SQL data types. It is returned because DB2 CLI is not currently able to determine if a column is updateable. Future versions of DB2 CLI for Unix and Windows servers will be able to determine if a column is updateable. • SQL_ATTR_READONLY is returned if the column is obtained from a catalog function call. Although DB2 CLI does not return them, ODBC also defines the following value: • SQL_ATTR_WRITE

This function is an extensible alternative to `SQLDescribeCol()`. `SQLDescribeCol()` returns a fixed set of descriptor information based on ANSI-89 SQL. `SQLColAttribute()` allows access to the more extensive set of descriptor information available in ANSI SQL-92 and DBMS vendor extensions.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 24. SQLColAttribute SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The buffer **CharacterAttributePtr* was not large enough to return the entire string value, so the string was truncated. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
07005	The statement did not return a result set.	The statement associated with the StatementHandle did not return a result set. There were no columns to describe.
07009	Invalid descriptor index.	The value specified for *ColumnNumber* was equal to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF. The value specified for the argument *ColumnNumber* was greater than the number of columns in the result set.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle.* An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()` or `SQLExecDirect()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* was less than 0.
HY091	Invalid descriptor field identifier.	The value specified for the argument *FieldIdentifier* was not one of the defined values, and was not an implementation-defined value.
HYC00	Driver not capable.	The value specified for the argument *FieldIdentifier* was not supported by DB2 CLI.

SQLColAttribute() can return any SQLSTATE that can be returned by SQLPrepare() or SQLExecute() when called after SQLPrepare() and before SQLExecute() depending on when the data source evaluates the SQL statement associated with the *StatementHandle*.

For performance reasons, an application should not call SQLColAttribute() before executing a statement.

Restrictions:

None.

Example:

```
/* get display size for column */
cliRC = SQLColAttribute(hstmt,
                        (SQLSMALLINT)(i + 1),
                        SQL_DESC_DISPLAY_SIZE,
                        NULL,
                        0,
                        NULL,
                        &colDataDisplaySize)
```

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Data type display (CLI) table" on page 393
- "Data type length (CLI) table" on page 391
- "Data type precision (CLI) table" on page 389
- "Data type scale (CLI) table" on page 390
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLCancel function (CLI) - Cancel statement" on page 49
- "SQLDescribeCol function (CLI) - Return a set of attributes for a column" on page 82
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbread.c -- How to read data from tables"
- "utilcli.c -- Utility functions used by DB2 CLI samples"

SQLColAttributes function (CLI) - Get column attributes

Deprecated:

Note:

In ODBC 3.0, `SQLColAttributes()` has been deprecated and replaced with `SQLColAttribute()`.

Although this version of DB2 CLI continues to support `SQLColAttributes()`, we recommend that you use `SQLColAttribute()` in your DB2 CLI programs so that they conform to the latest standards.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLColAttributesW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Migrating to the new function

The statement:

```
SQLColAttributes (hstmt, colNum, SQL_DESC_COUNT, NULL, len,
                  NULL, &numCols);
```

for example, would be rewritten using the new function as:

```
SQLColAttribute (hstmt, colNum, SQL_DESC_COUNT, NULL, len,
                 NULL, &numCols);
```

Related concepts:

- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLColAttribute function (CLI) - Return a column attribute" on page 54

SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

`SQLColumnPrivileges()` returns a list of columns and associated privileges for the specified table. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated from a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLColumnPrivilegesW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

SQLColumnPrivileges

```
SQLRETURN SQLColumnPrivileges(
              SQLHSTMT          StatementHandle,   /* hstmt */
              SQLCHAR           *CatalogName,      /* szCatalogName */
              SQLSMALLINT       NameLength1,       /* cbCatalogName */
              SQLCHAR           *SchemaName,       /* szSchemaName */
              SQLSMALLINT       NameLength2,       /* cbSchemaName */
              SQLCHAR           *TableName         /* szTableName */
              SQLSMALLINT       NameLength3,       /* cbTableName */
              SQLCHAR           *ColumnName,       /* szColumnName */
              SQLSMALLINT       NameLength4);      /* cbColumnName */
```

Function arguments:

Table 25. SQLColumnPrivileges arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR *	*CatalogName*	input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	input	Schema qualifier of table name.
SQLSMALLINT	*NameLength2*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*TableName*	input	Table name.
SQLSMALLINT	*NameLength3*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.
SQLCHAR *	*ColumnName*	input	Buffer that might contain a *pattern value* to qualify the result set by column name.
SQLSMALLINT	*NameLength4*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *ColumnName*, or SQL_NTS if *ColumnName* is null-terminated.

Usage:

The results are returned as a standard result set containing the columns listed in "Columns Returned by SQLColumnPrivileges" on page 65. The result set is ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME, and PRIVILEGE. If multiple privileges are associated with any given column, each privilege is returned as a separate row. A typical application might want to call this function after a call to `SQLColumns()` to determine column privilege information. The application should use the character strings returned in the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME columns of the `SQLColumns()` result set as input arguments to this function.

Since calls to SQLColumnPrivileges() in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating the calls.

Note that the *ColumnName* input argument accepts a search pattern, however, all other input arguments do not.

Sometimes, an application calls the function and no attempt is made to restrict the result set returned. For some data sources that contain a large quantity of tables, views and aliases for example, this scenario maps to an extremely large result set and very long retrieval times. In order to help reduce the long retrieval times, the configuration keyword SchemaList can be specified in the CLI initialization file to help restrict the result set when the application has supplied a null pointer for SchemaName. If the application specifies a SchemaName string, the SchemaList keyword is still used to restrict the output. Therefore, if the schema name supplied is not in the SchemaList string, then the result will be an empty result set.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns returned by SQLColumnPrivileges

Column 1 TABLE_CAT (VARCHAR(128) Data type)
Name of the catalog. The value is NULL if this table does not have catalogs.

Column 2 TABLE_SCHEM (VARCHAR(128))
Name of the schema containing TABLE_NAME.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the table or view.

Column 4 COLUMN_NAME (VARCHAR(128) not NULL)
Name of the column of the specified table or view.

Column 5 GRANTOR (VARCHAR(128))
Authorization ID of the user who granted the privilege.

Column 6 GRANTEE (VARCHAR(128))
Authorization ID of the user to whom the privilege is granted.

Column 7 PRIVILEGE (VARCHAR(128))
The column privilege. This can be:
- INSERT
- REFERENCES
- SELECT
- UPDATE

Note: Some IBM RDBMSs do not offer column level privileges at the column level. DB2 Database for Linux, UNIX, and Windows, DB2 for MVS/ESA™, and DB2 Server for VSE & VM support the UPDATE column privilege; there is one row in this result set for each updateable column. For all other privileges for DB2 Database for Linux, UNIX, and Windows, DB2 for MVS/ESA, and DB2 Server for VSE & VM, and for all privileges for other IBM RDBMSs, if a privilege has been granted at the table level, a row is present in this result set.

Column 8 IS_GRANTABLE (VARCHAR(3) Data type)
Indicates whether the grantee is permitted to grant the privilege to other users.

Either "YES" or "NO".

Note: The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents and order are identical to those defined for the `SQLColumnPrivileges()` result set in ODBC.

If there is more than one privilege associated with a column, then each privilege is returned as a separate row in the result set.

Return Codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 26. SQLColumnPrivileges SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40001	Serialization failure	The transaction was rolled back due to a resource deadlock with another transaction.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	*TableName* is NULL.
HY010	Function sequence error	An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()`, `SQLExecDirect()`, or `SQLSetPos()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
cliRC = SQLColumnPrivileges(hstmt,
                            NULL,
                            0,
                            tbSchema,
                            SQL_NTS,
                            tbName,
                            SQL_NTS,
                            colNamePattern,
                            SQL_NTS);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLColumns function (CLI) - Get column information for a table" on page 67
- "SQLTables function (CLI) - Get table information" on page 314
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SchemaList CLI/ODBC configuration keyword" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbinfo.c -- How to get information about tables from the system catalog tables"

SQLColumns function (CLI) - Get column information for a table

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLColumns()` returns a list of columns in the specified tables. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to fetch a result set generated by a query.

Unicode Equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLColumnsW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLColumns       (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR           *SchemaName,       /* szSchemaName */
```

```
SQLSMALLINT       NameLength2,       /* cbSchemaName */
SQLCHAR           *TableName,        /* szTableName */
SQLSMALLINT       NameLength3,       /* cbTableName */
SQLCHAR           *ColumnName,       /* szColumnName */
SQLSMALLINT       NameLength4);      /* cbColumnName */
```

Function arguments:

Table 27. SQLColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR *	*CatalogName*	input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	input	Buffer that might contain a *pattern value* to qualify the result set by schema name.
SQLSMALLINT	*NameLength2*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*TableName*	input	Buffer that might contain a *pattern value* to qualify the result set by table name.
SQLSMALLINT	*NameLength3*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.
SQLCHAR *	*ColumnName*	input	Buffer that might contain a *pattern value* to qualify the result set by column name.
SQLSMALLINT	*NameLength4*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *ColumnName*, or SQL_NTS if *ColumnName* is null-terminated.

Usage:

This function is called to retrieve information about the columns of either a table or a set of tables. An application might want to call this function after a call to `SQLTables()` to determine the columns of a table. The application should use the character strings returned in the TABLE_SCHEMA and TABLE_NAME columns of the `SQLTables()` result set as input to this function.

`SQLColumns()` returns a standard result set, ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and ORDINAL_POSITION. "Columns returned by SQLColumns" on page 69 lists the columns in the result set.

The *SchemaName*, *TableName*, and *ColumnName* input arguments accept search patterns.

Sometimes, an application calls the function and no attempt is made to restrict the result set returned. For some data sources that contain a large quantity of tables, views and aliases for example, this scenario maps to an extremely large result set and very long retrieval times. In order to help reduce the long retrieval times, the configuration keyword SchemaList can be specified in the CLI initialization file to help restrict the result set when the application has supplied a null pointer for SchemaName. If the application specifies a SchemaName string, the SchemaList keyword is still used to restrict the output. Therefore, if the schema name supplied is not in the SchemaList string, then the result will be an empty result set.

This function does not return information on the columns of a result set; `SQLDescribeCol()` or `SQLColAttribute()` should be used instead.

If the SQL_ATTR_LONGDATA_COMPAT attribute is set to SQL_LD_COMPAT_YES via either a call to `SQLSetConnectAttr()` or by setting the LONGDATACOMPAT keyword in the DB2 CLI initialization file, then the LOB data types are reported as SQL_LONGVARCHAR, SQL_LONGVARBINARY or SQL_LONGVARGRAPHIC.

Since calls to `SQLColumns()` in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_OWNER_SCHEMA_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Optimize SQL columns keyword and attribute:

It is possible to set up the DB2 CLI/ODBC Driver to optimize calls to `SQLColumns()` using either:
- OPTIMIZESQLCOLUMNS DB2 CLI/ODBC configuration keyword
- SQL_ATTR_OPTIMIZESQLCOLUMNS connection attribute of SQLSetConnectAttr()

If either of these values are set, then the information contained in the following columns will not be returned:
- Column 12 REMARKS
- Column 13 COLUMN_DEF

Columns returned by SQLColumns

Column 1 TABLE_CAT (VARCHAR(128))
Name of the catalog. The value is NULL if this table does not have catalogs.

Column 2 TABLE_SCHEM (VARCHAR(128))
Name of the schema containing TABLE_NAME.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the table, view, alias, or synonym.

Column 4 COLUMN_NAME (VARCHAR(128) not NULL)
Column identifier. Name of the column of the specified table, view, alias, or synonym.

Column 5 DATA_TYPE (SMALLINT not NULL)
SQL data type of column identified by COLUMN_NAME. This is one of the values in the Symbolic SQL Data Type column in the table of symbolic and default data types for CLI.

Column 6 TYPE_NAME (VARCHAR(128) not NULL)
Character string representing the name of the data type corresponding to DATA_TYPE.

Column 7 COLUMN_SIZE (INTEGER)
If the DATA_TYPE column value denotes a character or binary string, then this column contains the maximum length in SQLCHAR or SQLWCHAR elements for the column.

For date, time, timestamp data types, this is the total number of SQLCHAR or SQLWCHAR elements required to display the value when converted to character.

For numeric data types, this is either the total number of digits, or the total number of bits allowed in the column, depending on the value in the NUM_PREC_RADIX column in the result set.

For the XML data type, the length of zero is returned.

See also the table of data type precision.

Column 8 BUFFER_LENGTH (INTEGER)
The maximum number of bytes for the associated C buffer to store data from this column if SQL_C_DEFAULT were specified on the `SQLBindCol()`, `SQLGetData()` and `SQLBindParameter()` calls. This length does not include any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

See also the table of data type lengths.

Column 9 DECIMAL_DIGITS (SMALLINT)
The scale of the column. NULL is returned for data types where scale is not applicable.

See also the table of data type scale.

Column 10 NUM_PREC_RADIX (SMALLINT)
Either 10 or 2 or NULL. If DATA_TYPE is an approximate numeric data type, this column contains the value 2 and the COLUMN_SIZE column contains the number of bits allowed in the column.

If DATA_TYPE is an exact numeric data type, this column contains the value 10 and the COLUMN_SIZE contains the number of decimal digits allowed for the column.

For numeric data types, the DBMS can return a NUM_PREC_RADIX of 10 or 2.

NULL is returned for data types where the radix is not applicable.

Column 11 NULLABLE (SMALLINT not NULL)
SQL_NO_NULLS if the column does not accept NULL values.

SQL_NULLABLE if the column accepts NULL values.

Column 12 REMARKS (VARCHAR(254))
Might contain descriptive information about the column. It is possible that no information is returned in this column; see "Optimize SQL columns keyword and attribute" on page 69 for more details.

Column 13 COLUMN_DEF (VARCHAR(254))
The column's default value. If the default value is a numeric literal, then this column contains the character representation of the numeric literal with no enclosing single quotation marks. If the default value is a character string, then this column is that string enclosed in single quotation marks. If the default value a *pseudo-literal*, such as for DATE, TIME, and TIMESTAMP columns, then this column contains the keyword of the pseudo-literal (for example. CURRENT DATE) with no enclosing quotation marks.

If NULL was specified as the default value, then this column returns the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, then this column contains TRUNCATED with no enclosing single quotation marks. If no default value was specified, then this column is NULL.

It is possible that no information is returned in this column; see "Optimize SQL columns keyword and attribute" on page 69 for more details.

Column 14 SQL_DATA_TYPE (SMALLINT not NULL)
SQL data type, as it appears in the SQL_DESC_TYPE record field in the IRD. This column is the same as the DATA_TYPE column in "Columns returned by SQLColumns" on page 69 for the Date, Time, and Timestamp data types.

Column 15 SQL_DATETIME_SUB (SMALLINT)
The subtype code for datetime data types:
- SQL_CODE_DATE
- SQL_CODE_TIME
- SQL_CODE_TIMESTAMP

For all other data types this column returns NULL.

Column 16 CHAR_OCTET_LENGTH (INTEGER)
For single byte character sets, this is the same as COLUMN_SIZE. For the XML type, zero is returned. For all other data types it is NULL.

Column 17 ORDINAL_POSITION (INTEGER not NULL)
The ordinal position of the column in the table. The first column in the table is number 1.

Column 18 IS_NULLABLE (VARCHAR(254))
Contains the string 'NO' if the column is known to be not nullable; and 'YES' otherwise.

Note: This result set is identical to the X/Open CLI `Columns()` result set specification, which is an extended version of the `SQLColumns()` result set specified in ODBC V2. The ODBC `SQLColumns()` result set includes every column in the same position.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING

- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 28. SQLColumns SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restriction:

SQLColumns() does not support returning data from an alias of an alias. When called against an alias of an alias, SQLColumns() returns an empty result set.

Example:

```
/* get column information for a table */
cliRC = SQLColumns(hstmt,
                   NULL,
                   0,
                   tbSchemaPattern,
                   SQL_NTS,
                   tbNamePattern,
                   SQL_NTS,
                   colNamePattern,
                   SQL_NTS);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "XML data type" in *XML Guide*

Related tasks:

- "Retrieving query results in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*
- "Data conversions supported in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Data type length (CLI) table" on page 391
- "Data type precision (CLI) table" on page 389
- "Data type scale (CLI) table" on page 390
- "SQLColumnPrivileges function (CLI) - Get privileges associated with the columns of a table" on page 63
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "SQLTables function (CLI) - Get table information" on page 314

Related samples:

- "tbinfo.c -- How to get information about tables from the system catalog tables"

SQLConnect function (CLI) - Connect to a data source

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLConnect()` establishes a connection or a trusted connection to the target database. The application must supply a target SQL database, and optionally an authorization-name and an authentication-string.

A connection must be established before allocating a statement handle using `SQLAllocHandle()`.

Unicode Equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLConnectW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLConnect       (
               SQLHDBC           ConnectionHandle,       /* hdbc */
               SQLCHAR           *ServerName,             /* szDSN */
               SQLSMALLINT       ServerNameLength,       /* cbDSN */
               SQLCHAR           *UserName,               /* szUID */
               SQLSMALLINT       UserNameLength,         /* cbUID */
               SQLCHAR           *Authentication,         /* szAuthStr */
               SQLSMALLINT       AuthenticationLength);  /* cbAuthStr */
```

Function arguments:

Table 29. SQLConnect arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle
SQLCHAR *	*ServerName*	input	Data Source: The name or alias-name of the database.
SQLSMALLINT	*ServerNameLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *ServerName* argument.
SQLCHAR *	*UserName*	input	Authorization-name (user identifier)
SQLSMALLINT	*UserNameLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *UserName* argument.
SQLCHAR *	*Authentication*	input	Authentication-string (password)
SQLSMALLINT	*AuthenticationLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *Authentication* argument.

Usage:

The target database (also known as *data source*) for IBM RDBMSs is the database-alias. The application can obtain a list of databases available to connect to by calling `SQLDataSources()`.

The input length arguments to `SQLConnect()` (*ServerNameLength*, *UserNameLength*, *AuthenticationLength*) can be set to the actual length of their associated data in elements (SQLCHAR or SQLWCHAR), not including any null-terminating character, or to SQL_NTS to indicate that the associated data is null-terminated.

The *ServerName* and *UserName* argument values must not contain any blanks.

Stored procedures written using DB2 CLI must make a *null* `SQLConnect()` call. A null `SQLConnect()` is where the *ServerName*, *UserName*, and *Authentication* argument pointers are all set to NULL and their respective length arguments all set to 0. A null `SQLConnect()` still requires `SQLAllocHandle()` to be called first, but does not require that `SQLEndTran()` be called before `SQLDisconnect()`.

To create a trusted connection, specify the connection attribute SQL_ATTR_USE_TRUSTED_CONTEXT before calling `SQLConnect()`. If the database server accepts the connection as trusted the connection is treated as a trusted connection. Otherwise the connection is a regular connection and a warning is returned.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 30. SQLConnect SQLSTATEs

SQLSTATE	Description	Explanation
01679	Unable to establish a trusted connection.	DB2 CLI requested a trusted connection but the trust attributes of the connection do not match any trusted context object on the database server. The connection is allowed but it is a regular connection, not a trusted connection.
08001	Unable to connect to data source.	DB2 CLI was unable to establish a connection with the data source (server). The connection request was rejected because an existing connection established via embedded SQL already exists.
08002	Connection in use.	The specified *ConnectionHandle* has already been used to establish a connection with a data source and the connection is still open.
08004	The application server rejected establishment of the connection.	The data source (server) rejected the establishment of the connection.
28000	Invalid authorization specification.	The value specified for the argument *UserName* or the value specified for the argument *Authentication* violated restrictions defined by the data source.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument *ServerNameLength* was less than 0, but not equal to SQL_NTS and the argument *ServerName* was not a null pointer. The value specified for argument *UserNameLength* was less than 0, but not equal to SQL_NTS and the argument *UserName* was not a null pointer. The value specified for argument *AuthenticationLength* was less than 0, but not equal to SQL_NTS and the argument *Authentication* was not a null pointer.
HY501	Invalid data source name.	An invalid data source name was specified in argument *ServerName*.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

The implicit connection (or default database) option for IBM RDBMSs is not supported. `SQLConnect()` must be called before any SQL statements can be executed.

Example:

```
/* connect to the database */
cliRC = SQLConnect(hdbc,
                   (SQLCHAR *)db1Alias,
                   SQL_NTS,
                   (SQLCHAR *)user,
                   SQL_NTS,
                   (SQLCHAR *)pswd,
                   SQL_NTS);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Trusted connections through DB2 Connect" in *DB2 Connect User's Guide*

Related tasks:

- "Initializing CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Creating and terminating a trusted connection through CLI" in *DB2 Connect User's Guide*
- "Switching users on a trusted connection through CLI" in *DB2 Connect User's Guide*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLDataSources function (CLI) - Get list of data sources" on page 79
- "SQLDisconnect function (CLI) - Disconnect from a data source" on page 89
- "SQLDriverConnect function (CLI) - (Expanded) Connect to a data source" on page 91
- "SQLGetConnectAttr function (CLI) - Get current attribute setting" on page 146
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spserver.c -- Definition of various types of stored procedures"
- "dbconn.c -- How to connect to and disconnect from a database"
- "dbmcon.c -- How to use multiple databases"
- "dbmconx.c -- How to use multiple databases with embedded SQL."

SQLCopyDesc function (CLI) - Copy descriptor information between handles

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

`SQLCopyDesc()` copies descriptor information from one descriptor handle to another.

Syntax:

```
SQLRETURN   SQLCopyDesc (
               SQLHDESC       SourceDescHandle,   /* hDescSource */
               SQLHDESC       TargetDescHandle);  /* hDescTarget */
```

Function arguments:

Table 31. SQLCopyDesc arguments

Data type	Argument	Use	Description
SQLHDESC	*SourceDescHandle*	input	Source descriptor handle.
SQLHDESC	*TargetDescHandle*	input	Target descriptor handle. *TargetDescHandle* can be a handle to an application descriptor or an IPD. `SQLCopyDesc()` will return SQLSTATE HY016 (Cannot modify an implementation descriptor) if *TargetDescHandle* is a handle to an IRD.

Usage:

A call to `SQLCopyDesc()` copies the fields of the source descriptor handle to the target descriptor handle. Fields can only be copied to an application descriptor or an IPD, but not to an IRD. Fields can be copied from either an application or an implementation descriptor.

All fields of the descriptor, except SQL_DESC_ALLOC_TYPE (which specifies whether the descriptor handle was automatically or explicitly allocated), are copied, whether or not the field is defined for the destination descriptor. Copied fields overwrite the existing fields in the *TargetDescHandle*.

All descriptor fields are copied, even if *SourceDescHandle* and *TargetDescHandle* are on two different connections or environments.

The call to `SQLCopyDesc()` is immediately aborted if an error occurs.

When the SQL_DESC_DATA_PTR field is copied, a consistency check is performed. If the consistency check fails, SQLSTATE HY021 (Inconsistent descriptor information.) is returned and the call to `SQLCopyDesc()` is immediately aborted.

Note: Descriptor handles can be copied across connections or environments. An application may, however, be able to associate an explicitly allocated descriptor handle with a *StatementHandle*, rather than calling `SQLCopyDesc()` to copy fields from one descriptor to another. An explicitly allocated descriptor can be associated with another *StatementHandle* on the same *ConnectionHandle* by setting the SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC statement attribute to the handle of the explicitly allocated descriptor. When this is done, `SQLCopyDesc()` does not have to be called to copy descriptor field values from one descriptor to another.

A descriptor handle cannot be associated with a *StatementHandle* on another *ConnectionHandle*, however; to use the same descriptor field values on *StatementHandle* on different *ConnectionHandle*, `SQLCopyDesc()` has to be called.

Copying rows between tables

An ARD on one statement handle can serve as the APD on another statement handle. This allows an application to copy rows between tables without copying data at the application level. To do this, an application calls `SQLCopyDesc()` to copy the fields of an ARD that describes a fetched row of a table, to the APD for a parameter in an INSERT statement on another statement handle. The SQL_ACTIVE_STATEMENTS *InfoType* returned by the driver for a call to `SQLGetInfo()` must be greater than 1 for this operation to succeed.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

When `SQLCopyDesc()` returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, an associated SQLSTATE value may be obtained by calling `SQLGetDiagRec()` with a `HandleType` of SQL_HANDLE_DESC and a `Handle` of *TargetDescHandle*. If an invalid *SourceDescHandle* was passed in the call, SQL_INVALID_HANDLE will be returned, but no SQLSTATE will be returned.

When an error is returned, the call to `SQLCopyDesc()` is immediately aborted, and the contents of the fields in the *TargetDescHandle* descriptor are undefined.

Table 32. SQLCopyDesc SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was trying to connect failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY007	Associated statement is not prepared.	*SourceDescHandle* was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called.
HY016	Cannot modify an implementation row descriptor.	*TargetDescHandle* was associated with an IRD.

Table 32. SQLCopyDesc SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY021	Inconsistent descriptor information.	The descriptor information checked during a consistency check was not consistent.
HY092	Option type out of range.	The call to `SQLCopyDesc()` prompted a call to `SQLSetDescField()`, but **ValuePtr* was not valid for the *FieldIdentifier* argument on *TargetDescHandle*.

Restrictions:

None.

Example:

```
SQLHANDLE hIRD, hARD; /* descriptor handles */

/* ... */

/* copy descriptor information between handles */
rc = SQLCopyDesc(hIRD, hARD);
```

Related concepts:

- "Consistency checks for descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record" on page 173
- "SQLSetDescField function (CLI) - Set a single field of a descriptor record" on page 276
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbuse.c -- How to use a database"

SQLDataSources function (CLI) - Get list of data sources

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLDataSources()` returns a list of target databases available, one at a time. A database must be cataloged to be available.

`SQLDataSources()` is usually called before a connection is made, to determine the databases that are available to connect to.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLDataSourcesW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLDataSources (
               SQLHENV          EnvironmentHandle,  /* henv */
               SQLUSMALLINT     Direction,          /* fDirection */
               SQLCHAR          *ServerName,        /* *szDSN */
               SQLSMALLINT      BufferLength1,      /* cbDSNMax */
               SQLSMALLINT      *NameLength1Ptr,    /* *pcbDSN */
               SQLCHAR          *Description,       /* *szDescription */
               SQLSMALLINT      BufferLength2,      /* cbDescriptionMax */
               SQLSMALLINT      *NameLength2Ptr);   /* *pcbDescription */
```

Function arguments:

Table 33. SQLDataSources arguments

Data type	Argument	Use	Description
SQLHENV	*EnvironmentHandle*	input	Environment handle.
SQLUSMALLINT	*Direction*	input	Used by application to request the first data source name in the list or the next one in the list. *Direction* can take on only the following values: • SQL_FETCH_FIRST • SQL_FETCH_NEXT
SQLCHAR *	*ServerName*	output	Pointer to buffer in which to return the data source name.
SQLSMALLINT	*BufferLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *ServerName* buffer. This number should be less than or equal to SQL_MAX_DSN_LENGTH + 1.
SQLSMALLINT *	*NameLength1Ptr*	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in **ServerName*. If the number of SQLCHAR or SQLWCHAR elements available to return is greater than or equal to *BufferLength1*, the data source name in **ServerName* is truncated to *BufferLength1* minus the length of a null-termination character.
SQLCHAR *	*Description*	output	Pointer to buffer where the description of the data source is returned. DB2 CLI will return the ***Comment*** field associated with the database catalogued to the DBMS.
SQLSMALLINT	*BufferLength2*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *Description* buffer.

Table 33. SQLDataSources arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	*NameLength2Ptr*	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in **Description*. If the number of SQLCHAR or SQLWCHAR elements available to return is greater than or equal to *BufferLength2*, the driver description in **Description* is truncated to *BufferLength2* minus the length of a null-termination character.

Usage:

The application can call this function any time with *Direction* set to either SQL_FETCH_FIRST or SQL_FETCH_NEXT.

If SQL_FETCH_FIRST is specified, the first database in the list will always be returned.

If SQL_FETCH_NEXT is specified:
- Directly following a SQL_FETCH_FIRST call, the second database in the list is returned
- Before any other `SQLDataSources()` call, the first database in the list is returned
- When there are no more databases in the list, SQL_NO_DATA_FOUND is returned. If the function is called again, the first database is returned.
- Any other time, the next database in the list is returned.

In an ODBC environment, the ODBC Driver Manager will perform this function.

Since the IBM RDBMSs always returns the description of the data source blank padded to 30 bytes, DB2 CLI will do the same.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

Diagnostics:

Table 34. SQLDataSources SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data source name returned in the argument *ServerName* was longer than the value specified in the argument *BufferLength1*. The argument *NameLength1Ptr* contains the length of the full data source name. (Function returns SQL_SUCCESS_WITH_INFO.) The data source name returned in the argument *Description* was longer than the value specified in the argument *BufferLength2*. The argument *NameLength2Ptr* contains the length of the full data source description. (Function returns SQL_SUCCESS_WITH_INFO.)
58004	Unexpected system failure.	Unrecoverable system error.

Table 34. SQLDataSources SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by `SQLGetDiagRec()` in the *MessageText* argument describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument *BufferLength1* was less than 0. The value specified for argument *BufferLength2* was less than 0.
HY103	Direction option out of range.	The value specified for the argument *Direction* was not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

Authorization:

None.

Example:

```
/* get list of data sources */
cliRC = SQLDataSources(henv,
                       SQL_FETCH_FIRST,
                       dbAliasBuf,
                       SQL_MAX_DSN_LENGTH + 1,
                       &aliasLen,
                       dbCommentBuf,
                       255,
                       &commentLen);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Setting up the CLI environment" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLConnect function (CLI) - Connect to a data source" on page 73
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "ininfo.c -- How to get information at the instance level"

SQLDescribeCol function (CLI) - Return a set of attributes for a column

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLDescribeCol() returns a set of commonly used descriptor information (column name, type, precision, scale, nullability) for the indicated column in the result set generated by a query.

This information is also available in the fields of the IRD.

If the application needs only one attribute of the descriptor information, or needs an attribute not returned by SQLDescribeCol(), the SQLColAttribute() function can be used in place of SQLDescribeCol().

Either SQLPrepare() or SQLExecDirect() must be called before calling this function.

This function (or SQLColAttribute()) is usually called before a bind column function (SQLBindCol(), SQLBindFileToCol()) to determine the attributes of a column before binding it to an application variable.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLDescribeColW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLDescribeCol (
               SQLHSTMT          StatementHandle,    /* hstmt */
               SQLUSMALLINT      ColumnNumber,       /* icol */
               SQLCHAR           *ColumnName,        /* szColName */
               SQLSMALLINT       BufferLength,       /* cbColNameMax */
               SQLSMALLINT       *NameLengthPtr,     /* pcbColName */
               SQLSMALLINT       *DataTypePtr,       /* pfSqlType */
               SQLULEN           *ColumnSizePtr,     /* pcbColDef */
               SQLSMALLINT       *DecimalDigitsPtr,  /* pibScale */
               SQLSMALLINT       *NullablePtr);      /* pfNullable */
```

Function arguments:

Table 35. SQLDescribeCol arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLUSMALLINT	*ColumnNumber*	input	Column number to be described. Columns are numbered sequentially from left to right, starting at 1. This can also be set to 0 to describe the bookmark column.
SQLCHAR *	*ColumnName*	output	Pointer to column name buffer. This value is read from the SQL_DESC_NAME field of the IRD. This is set to NULL if the column name cannot be determined.
SQLSMALLINT	*BufferLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the * *ColumnName* buffer.

Table 35. SQLDescribeCol arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	*NameLengthPtr*	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in * *ColumnName*. Truncation of column name (* *ColumnName*) to *BufferLength* - 1 SQLCHAR or SQLWCHAR elements occurs if *NameLengthPtr* is greater than or equal to *BufferLength*.
SQLSMALLINT *	*DataTypePtr*	output	Base SQL data type of column. To determine if there is a User Defined Type associated with the column, call `SQLColAttribute()` with *fDescType* set to SQL_COLUMN_DISTINCT_TYPE. Refer to the Symbolic SQL Data Type column of the symbolic and default data types table for the data types that are supported.
SQLULEN *	*ColumnSizePtr*	output	Precision of column as defined in the database. If *fSqlType* denotes a graphic or DBCLOB SQL data type, then this variable indicates the maximum number of double-byte *characters* the column can hold.
SQLSMALLINT *	*DecimalDigitsPtr*	output	Scale of column as defined in the database (only applies to SQL_DECIMAL, SQL_NUMERIC, SQL_TIMESTAMP). Refer to the data type scale table for the scale of each of the SQL data types.
SQLSMALLINT *	*NullablePtr*	output	Indicates whether NULLS are allowed for this column • SQL_NO_NULLS • SQL_NULLABLE

Usage:

Columns are identified by a number, are numbered sequentially from left to right, and can be described in any order.

- Column numbers start at 1 if bookmarks are not used (SQL_ATTR_USE_BOOKMARKS statement attribute set to SQL_UB_OFF).
- The *ColumnNumber* argument can be set to 0 to describe the bookmark column if bookmarks are used (the statement attribute is set to SQL_UB_ON).

If a null pointer is specified for any of the pointer arguments, DB2 CLI assumes that the information is not needed by the application and nothing is returned.

If the column is a User Defined Type, `SQLDescribeCol()` only returns the built-in type in *DataTypePtr*. Call `SQLColAttribute()` with *fDescType* set to SQL_COLUMN_DISTINCT_TYPE to obtain the User Defined Type.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

If SQLDescribeCol() returns either SQL_ERROR, or SQL_SUCCESS_WITH_INFO, one of the following SQLSTATEs can be obtained by calling the SQLGetDiagRec() or SQLGetDiagField() function.

Table 36. SQLDescribeCol SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The column name returned in the argument * *ColumnName* was longer than the value specified in the argument *BufferLength*. The argument * *NameLengthPtr* contains the length of the full column name. (Function returns SQL_SUCCESS_WITH_INFO.)
07005	The statement did not return a result set.	The statement associated with the *StatementHandle* did not return a result set. There were no columns to describe. (Call SQLNumResultCols() first to determine if there are any rows in the result set.)
07009	Invalid descriptor index	The value specified for *ColumnNumber* was equal to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF. The value specified for the argument *ColumnNumber* was greater than the number of columns in the result set.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, SQLCancel() was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called prior to calling SQLPrepare() or SQLExecDirect() for the *StatementHandle*. The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The length specified in argument *BufferLength* less than 1.
HYC00	Driver not capable.	The SQL data type of column *ColumnNumber* is not recognized by DB2 CLI.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

Restrictions:

The following ODBC defined data types are not supported:
- SQL_BIT
- SQL_TINYINT

Example:

```
/* return a set of attributes for a column */
cliRC = SQLDescribeCol(hstmt,
                       (SQLSMALLINT)(i + 1),
                       colName,
                       sizeof(colName),
                       &colNameLen,
                       &colType,
                       &colSize,
                       &colScale,
                       NULL);
```

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLColAttribute function (CLI) - Return a column attribute" on page 54
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLNumResultCols function (CLI) - Get number of result columns" on page 237
- "SQLPrepare function (CLI) - Prepare a statement" on page 242
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbread.c -- How to read data from tables"

SQLDescribeParam function (CLI) - Return description of a parameter marker

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 1.0	**ISO CLI**

`SQLDescribeParam()` returns the description of a parameter marker associated with a prepared SQL statement. This information is also available in the fields of the IPD. If deferred prepared is enabled, and this is the first call to `SQLDescribeParam()`, `SQLNumResultCols()`, or `SQLDescribeCol()`, the call will force a PREPARE of the SQL statement to be flowed to the server.

Syntax:

```
SQLRETURN   SQLDescribeParam (
               SQLHSTMT          StatementHandle,     /* hstmt */
               SQLUSMALLINT      ParameterNumber,     /* ipar */
               SQLSMALLINT       *DataTypePtr,        /* pfSqlType */
               SQLULEN           *ParameterSizePtr,   /* pcbParamDef */
               SQLSMALLINT       *DecimalDigitsPtr,   /* pibScale */
               SQLSMALLINT       *NullablePtr);       /* pfNullable */
```

Function arguments:

Table 37. SQLDescribeParam arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*ParameterNumber*	input	Parameter marker number ordered sequentially in increasing parameter order, starting at 1.
SQLSMALLINT *	*DataTypePtr*	output	Pointer to a buffer in which to return the SQL data type of the parameter. This value is read from the SQL_DESC_CONCISE_TYPE record field of the IPD. When ColumnNumber is equal to 0 (for a bookmark column), SQL_BINARY is returned in **DataTypePtr* for variable-length bookmarks.
SQLULEN *	*ParameterSizePtr*	output	Pointer to a buffer in which to return the size of the column or expression of the corresponding parameter marker as defined by the data source.
SQLSMALLINT *	*DecimalDigitsPtr*	output	Pointer to a buffer in which to return the number of decimal digits of the column or expression of the corresponding parameter as defined by the data source.
SQLSMALLINT *	*NullablePtr*	output	Pointer to a buffer in which to return a value that indicates whether the parameter allows NULL values. This value is read from the SQL_DESC_NULLABLE field of the IPD. One of the following: • SQL_NO_NULLS: The parameter does not allow NULL values (this is the default value). • SQL_NULLABLE: The parameter allows NULL values. • SQL_NULLABLE_UNKNOWN: Cannot determine if the parameter allows NULL values.

Usage:

Parameter markers are numbered in increasing order as they appear in the SQL statement, starting with 1.

`SQLDescribeParam()` does not return the type (input, input/output, or output) of a parameter in an SQL statement. Except in calls to stored procedures, all parameters in SQL statements are input parameters. To determine the type of each parameter in a call to a stored procedure, call `SQLProcedureColumns()`.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 38. SQLDescribeParam SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)

Table 38. SQLDescribeParam SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07009	Invalid descriptor index.	The value specified for the argument *ParameterNumber* less than 1. The value specified for the argument *ParameterNumber* was greater than the number of parameters in the associated SQL statement. The parameter marker was part of a non-DML statement. The parameter marker was part of a SELECT list.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
21S01	Insert value list does not match column list.	The number of parameters in the INSERT statement did not match the number of columns in the table named in the statement.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()` `SQLExecDirect()`, `SQLBulkOperations()`, or `SQLSetPos()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	The function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HYC00	Driver not capable.	The schema function stored procedures are not accessible on the server. Install the schema function stored procedures on the server and ensure they are accessible.

Restrictions:

None.

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Parameter marker binding in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLCancel function (CLI) - Cancel statement" on page 49
- "SQLExecute function (CLI) - Execute a statement" on page 106
- "SQLNumParams function (CLI) - Get number of parameters in a SQL statement" on page 233
- "SQLPrepare function (CLI) - Prepare a statement" on page 242
- "SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure" on page 251
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLDisconnect function (CLI) - Disconnect from a data source

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLDisconnect()` closes the connection associated with the database connection handle.

`SQLEndTran()` must be called before calling `SQLDisconnect()` if an outstanding transaction exists on this connection.

After calling this function, either call `SQLConnect()` to connect to another database, or use `SQLFreeHandle()` to free the connection handle.

Syntax:

```
SQLRETURN   SQLDisconnect (SQLHDBC        ConnectionHandle;) /* hdbc */
```

Function arguments:

Table 39. SQLDisconnect arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle

Usage:

If an application calls `SQLDisconnect()` before it has freed all the statement handles associated with the connection, DB2 CLI frees them after it successfully disconnects from the database.

If SQL_SUCCESS_WITH_INFO is returned, it implies that even though the disconnect from the database is successful, additional error or implementation specific information is available. For example, a problem was encountered on the clean up subsequent to the disconnect, or if there is no current connection because of an event that occurred independently of the application (such as communication failure).

After a successful SQLDisconnect() call, the application can re-use *ConnectionHandle* to make another SQLConnect() or SQLDriverConnect() request.

An application should not rely on SQLDisconnect() to close cursors (with both stored procedures and regular client applications). In both cases the cursor should be closed using SQLCloseCursor(), then the statement handle freed using SQLFreeHandle().

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 40. SQLDisconnect SQLSTATEs

SQLSTATE	Description	Explanation
01002	Disconnect error.	An error occurred during the disconnect. However, the disconnect succeeded. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	The connection specified in the argument *ConnectionHandle* was not open.
25000 **25**501	Invalid transaction state.	There was a transaction in process on the connection specified by the argument *ConnectionHandle*. The transaction remains active, and the connection cannot be disconnected. **Note:** This error does not apply to stored procedures written in DB2 CLI.
25501	Invalid transaction state.	There was a transaction in process on the connection specified by the argument *ConnectionHandle*. The transaction remains active, and the connection cannot be disconnected.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Example:

```
SQLHANDLE hdbc; /* connection handle */

/* ... */

/* disconnect from the database */
  cliRC = SQLDisconnect(hdbc);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Terminating a CLI application" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLConnect function (CLI) - Connect to a data source" on page 73
- "SQLDriverConnect function (CLI) - (Expanded) Connect to a data source" on page 91
- "SQLEndTran function (CLI) - End transactions of a connection or an Environment" on page 97
- "SQLFreeHandle function (CLI) - Free handle resources" on page 140
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spserver.c -- Definition of various types of stored procedures"
- "dbconn.c -- How to connect to and disconnect from a database"
- "dbmcon.c -- How to use multiple databases"
- "dbmconx.c -- How to use multiple databases with embedded SQL."

SQLDriverConnect function (CLI) - (Expanded) Connect to a data source

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLDriverConnect()` is an alternative to `SQLConnect()`. Both functions establish a connection to the target database, but `SQLDriverConnect()` supports additional connection parameters and the ability to prompt the user for connection information.

Use `SQLDriverConnect()` when the data source requires parameters other than the 3 input arguments supported by `SQLConnect()` (data source name, user ID and password), or when you want to use DB2 CLI's graphical user interface to prompt the user for mandatory connection information.

Once a connection is established, the completed connection string is returned. Applications can store this string for future connection requests.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLDriverConnectW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

Generic

```
SQLRETURN  SQLDriverConnect (
        SQLHDBC       ConnectionHandle,              /* hdbc */
        SQLHWND       WindowHandle,                  /* hwnd */
        SQLCHAR       *InConnectionString,           /* szConnStrIn */
        SQLSMALLINT   InConnectionStringLength,      /* cbConnStrIn */
        SQLCHAR       *OutConnectionString,          /* szConnStrOut */
        SQLSMALLINT   OutConnectionStringCapacity,   /* cbConnStrOutMax */
        SQLSMALLINT   *OutConnectionStringLengthPtr, /* pcbConnStrOut */
        SQLUSMALLINT  DriverCompletion);             /* fDriverCompletion */
```

Function arguments:

Table 41. SQLDriverConnect arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle
SQLHWND	*WindowHandle*	input	Window handle. On the Windows platform, this is the parent Windows handle. Currently the window handle is only supported on Windows. If a NULL is passed, then no dialog will be presented.
SQLCHAR *	*InConnectionString*	input	A full, partial or empty (null pointer) connection string (see syntax and description below).
SQLSMALLINT	*InConnectionStringLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *InConnectionString*.
SQLCHAR *	*OutConnectionString*	output	Pointer to buffer for the completed connection string. If the connection was established successfully, this buffer will contain the completed connection string. Applications should allocate at least SQL_MAX_OPTION_STRING_LENGTH bytes for this buffer.
SQLSMALLINT	*OutConnectionString Capacity*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *OutConnectionString*.
SQLCHAR *	*OutConnectionString LengthPtr*	output	Pointer to the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in the *OutConnectionString* buffer. If the value of **OutConnectionStringLengthPtr* is greater than or equal to *OutConnectionStringCapacity*, the completed connection string in *OutConnectionString* is truncated to *OutConnectionStringCapacity* - 1 SQLCHAR or SQLWCHAR elements.
SQLUSMALLINT	*DriverCompletion*	input	Indicates when DB2 CLI should prompt the user for more information. Possible values: • SQL_DRIVER_PROMPT • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE_REQUIRED • SQL_DRIVER_NOPROMPT

Usage:

InConnectionString Argument

A request connection string has the following syntax:

connection-string ::= attribute[;] | attribute; connection-string

attribute ::= attribute-keyword=attribute-value
| DRIVER=[{]attribute-value[}]

attribute-keyword ::= DSN | UID | PWD | NEWPWD
| driver-defined-attribute-keyword

attribute-value ::= character-string
driver-defined-attribute-keyword ::= identifier

where
- character-string has zero or more SQLCHAR or SQLWCHAR elements
- identifier has one or more SQLCHAR or SQLWCHAR elements
- attribute-keyword is case insensitive
- attribute-value may be case sensitive
- the value of the **DSN** keyword does not consist solely of blanks
- NEWPWD is used as part of a change password request. The application can either specify the new string to use, for example, NEWPWD=anewpass; or specify NEWPWD=; and rely on a dialog box generated by the DB2 CLI driver to prompt for the new password

Because of connection string and initialization file grammar, keywords and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (\) character. For DB2 CLI Version 2, braces are required around the DRIVER keyword.

If any keywords are repeated in the browse request connection string, DB2 CLI uses the value associated with the first occurrence of the keyword. If the **DSN** and **DRIVER** keywords are included in the same browse request connection string, DB2 CLI uses whichever keyword appears first.

OutConnectionString Argument

The result connection string is a list of connection attributes. A connection attribute consists of an attribute keyword and a corresponding attribute value. The browse result connection string has the following syntax:

connection-string ::= attribute[;] | attribute; connection-string

attribute ::= [*]attribute-keyword=attribute-value
attribute-keyword ::= ODBC-attribute-keyword
| driver-defined-attribute-keyword

ODBC-attribute-keyword = {UID | PWD}[:localized-identifier]
driver-defined-attribute-keyword ::= identifier[:localized-identifier]

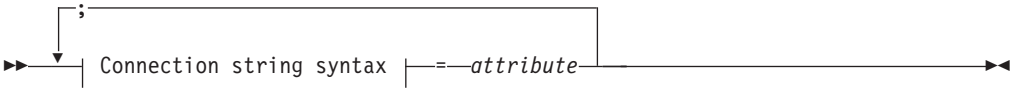
attribute-value ::= {attribute-value-list} | ?
(The braces are literal; they are returned by DB2 CLI.)

attribute-value-list ::= character-string [:localized-character string] | character-string [:localized-character string], attribute-value-list

where
- character-string and localized-character string have zero or more SQLCHAR or SQLWCHAR elements
- identifier and localized-identifier have one or more SQLCHAR or SQLWCHAR elements; attribute-keyword is case insensitive
- attribute-value may be case sensitive

Because of connection string and initialization file grammar, keywords, localized identifiers, and attribute values that contain the characters **[]{}(),;?*=!@** should be avoided. Because of the grammar in the system information, keywords and data source names cannot contain the backslash (\) character.

The connection string is used to pass one or more values needed to complete a connection. The contents of the connection string and the value of *DriverCompletion* will determine if DB2 CLI needs to establish a dialog with the user.



Connection string syntax

Each keyword above has an attribute that is equal to the following:

DSN Data source name. The name or alias-name of the database. Required if *DriverCompletion* is equal to SQL_DRIVER_NOPROMPT.

UID Authorization-name (user identifier).

PWD The password corresponding to the authorization name. If there is no password for the user ID, an empty value is specified (PWD=;).

NEWPWD
New password used as part of a change password request. The application can either specify the new string to use, for example, NEWPWD=anewpass; or specify NEWPWD=; and rely on a dialog box generated by the DB2 CLI driver to prompt for the new password (set the *DriverCompletion* argument to anything other than SQL_DRIVER_NOPROMPT).

Any one of the CLI keywords can be specified on the connection string. If any keywords are repeated in the connection string, the value associated with the first occurrence of the keyword is used.

If any keywords exists in the CLI initialization file, the keywords and their respective values are used to augment the information passed to DB2 CLI in the

connection string. If the information in the CLI initialization file contradicts information in the connection string, the values in connection string take precedence.

If the end user *Cancel*s a dialog box presented, SQL_NO_DATA_FOUND is returned.

The following values of *DriverCompletion* determines when a dialog will be opened:

SQL_DRIVER_PROMPT:
A dialog is always initiated. The information from the connection string and the CLI initialization file are used as initial values, to be supplemented by data input via the dialog box.

SQL_DRIVER_COMPLETE:
A dialog is only initiated if there is insufficient information in the connection string. The information from the connection string is used as initial values, to be supplemented by data entered via the dialog box.

SQL_DRIVER_COMPLETE_REQUIRED:
A dialog is only initiated if there is insufficient information in the connection string. The information from the connection string is used as initial values. Only mandatory information is requested. The user is prompted for required information only.

SQL_DRIVER_NOPROMPT:
The user is not prompted for any information. A connection is attempted with the information contained in the connection string. If there is not enough information, SQL_ERROR is returned.

Once a connection is established, the complete connection string is returned. Applications that need to set up multiple connections to the same database for a given user ID should store this output connection string. This string can then be used as the input connection string value on future `SQLDriverConnect()` calls.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA_FOUND
- SQL_INVALID_HANDLE
- SQL_ERROR

Diagnostics:

All of the diagnostics generated by `SQLConnect()` can be returned here as well. The following table shows the additional diagnostics that can be returned.

Table 42. SQLDriverConnect SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The buffer *szConnstrOut* was not large enough to hold the entire connection string. The argument **OutConnectionStringLengthPtr* contains the actual length of the connection string available for return. (Function returns SQL_SUCCESS_WITH_INFO)

Table 42. SQLDriverConnect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
01S00	Invalid connection string attribute.	An invalid keyword or attribute value was specified in the input connection string, but the connection to the data source was successful anyway because one of the following occurred: • The unrecognized keyword was ignored. • The invalid attribute value was ignored, the default value was used instead. (Function returns SQL_SUCCESS_WITH_INFO)
HY000	General error. Dialog Failed	The information specified in the connection string was insufficient for making a connect request, but the dialog was prohibited by setting *fCompletion* to SQL_DRIVER_NOPROMPT. The attempt to display the dialog failed.
HY090	Invalid string or buffer length.	The value specified for *InConnectionStringLength* was less than 0, but not equal to SQL_NTS. The value specified for *OutConnectionStringCapacity* was less than 0.
HY110	Invalid driver completion.	The value specified for the argument *fCompletion* was not equal to one of the valid values.

Restrictions:

None.

Example:

```
rc = SQLDriverConnect(hdbc,
                      (SQLHWND)sqlHWND,
                      InConnectionString,
                      InConnectionStringLength,
                      OutConnectionString,
                      OutConnectionStringCapacity,
                      StrLength2,
                      DriveCompletion);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Initializing CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLConnect function (CLI) - Connect to a data source" on page 73
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbcongui.c -- How to connect to a database with a graphical user interface (GUI)"
- "dbconn.c -- How to connect to and disconnect from a database"

SQLEndTran function (CLI) - End transactions of a connection or an Environment

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLEndTran()` requests a commit or rollback operation for all active operations on all statements associated with a connection, or for all connections associated with an environment.

Syntax:

```
SQLRETURN   SQLEndTran (
               SQLSMALLINT       HandleType,       /* fHandleType */
               SQLHANDLE         Handle,           /* hHandle */
               SQLSMALLINT       CompletionType);  /* fType */
```

Function arguments:

Table 43. SQLEndTran arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	*Handle* type identifier. Contains either SQL_HANDLE_ENV if *Handle* is an environment handle, or SQL_HANDLE_DBC if *Handle* is a connection handle.
SQLHANDLE	*Handle*	input	The handle, of the type indicated by *HandleType*, indicating the scope of the transaction. See the "Usage" section below for more information.
SQLSMALLINT	*CompletionType*	input	One of the following two values: • SQL_COMMIT • SQL_ROLLBACK

Usage:

If *HandleType* is SQL_HANDLE_ENV and *Handle* is a valid environment handle, then DB2 CLI will attempt to commit or roll back transactions one at a time, depending on the value of *CompletionType*, on all connections that are in a connected state on that environment. SQL_SUCCESS will only be returned if it receives SQL_SUCCESS for each connection. If it receives SQL_ERROR on one or more connections, it will return SQL_ERROR to the application, and the diagnostic information will be placed in the diagnostic data structure of the environment. To determine which connection(s) failed during the commit or rollback operation, the application can call `SQLGetDiagRec()` for each connection.

`SQLEndTran()` should not be used when working in a Distributed Unit of Work environment. The transaction manager APIs should be used instead.

If *CompletionType* is SQL_COMMIT, `SQLEndTran()` issues a commit request for all active operations on any statement associated with an affected connection. If *CompletionType* is SQL_ROLLBACK, `SQLEndTran()` issues a rollback request for all active operations on any statement associated with an affected connection. If no transactions are active, `SQLEndTran()` returns SQL_SUCCESS with no effect on any data sources.

To determine how transaction operations affect cursors, an application calls SQLGetInfo() with the SQL_CURSOR_ROLLBACK_BEHAVIOR and SQL_CURSOR_COMMIT_BEHAVIOR options.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_DELETE, SQLEndTran() closes and deletes all open cursors on all statements associated with the connection and discards all pending results. SQLEndTran() leaves any statement present in an allocated (unprepared) state; the application can reuse them for subsequent SQL requests or can call SQLFreeStmt() or SQLFreeHandle() with a *HandleType* of SQL_HANDLE_STMT to deallocate them.

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_CLOSE, SQLEndTran() closes all open cursors on all statements associated with the connection. SQLEndTran() leaves any statement present in a prepared state; the application can call SQLExecute() for a statement associated with the connection without first calling SQLPrepare().

If the SQL_CURSOR_ROLLBACK_BEHAVIOR or SQL_CURSOR_COMMIT_BEHAVIOR value equals SQL_CB_PRESERVE, SQLEndTran() does not affect open cursors associated with the connection. Cursors remain at the row they pointed to prior to the call to SQLEndTran().

When autocommit mode is off, calling SQLEndTran() with either SQL_COMMIT or SQL_ROLLBACK when no transaction is active will return SQL_SUCCESS (indicating that there is no work to be committed or rolled back) and have no effect on the data source, unless errors not related to transactions occur.

When autocommit mode is on, calling SQLEndTran() with a *CompletionType* of either SQL_COMMIT or SQL_ROLLBACK always returns SQL_SUCCESS, unless errors not related to transactions occur.

When a DB2 CLI application is running in autocommit mode, the DB2 CLI driver does not pass the SQLEndTran() statement to the server.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 44. SQLEndTran SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	The *ConnectionHandle* was not in a connected state.
08007	Connection failure during transaction.	The connection associated with the *ConnectionHandle* failed during the execution of the function and it cannot be determined whether the requested **COMMIT** or **ROLLBACK** occurred before the failure.
40001	Transaction rollback.	The transaction was rolled back due to a resource deadlock with another transaction.

Table 44. SQLEndTran SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	An asynchronously executing function was called for a *StatementHandle* associated with the *ConnectionHandle* and was still executing when `SQLEndTran()` was called. `SQLExecute()` or `SQLExecDirect()` was called for a *StatementHandle* associated with the *ConnectionHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY012	Invalid transaction code.	The value specified for the argument *CompletionType* was neither SQL_COMMIT nor SQL_ROLLBACK.
HY092	Option type out of range.	The value specified for the argument *HandleType* was neither SQL_HANDLE_ENV nor SQL_HANDLE_DBC.

Restrictions:

None.

Example:

```
/* commit all active transactions on the connection */
cliRC = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT)

/* ... */

/* rollback all active transactions on the connection */
cliRC = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK);

/* ... */

/* rollback all active transactions on all connections
   in this environment */
cliRC = SQLEndTran(SQL_HANDLE_ENV, henv, SQL_ROLLBACK);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Multisite updates (two phase commit) in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Terminating a CLI application" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLFreeHandle function (CLI) - Free handle resources" on page 140

- "SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record" on page 173
- "SQLGetInfo function (CLI) - Get general information" on page 180
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:
- "tbmod.c -- How to modify table data"
- "dbuse.c -- How to use a database"

SQLError function (CLI) - Retrieve error information

Deprecated:

Note:

In ODBC 3.0, `SQLError()` has been deprecated and replaced with `SQLGetDiagRec()` and `SQLGetDiagField()`.

Although this version of DB2 CLI continues to support `SQLError()`, we recommend that you use `SQLGetDiagRec()` in your DB2 CLI programs so that they conform to the latest standards.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLErrorW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Migrating to the new function

To read the error diagnostic records for a statement handle, the `SQLError()` function,

```
SQLError(henv, hdbc, hstmt, *szSqlState, *pfNativeError,
         *szErrorMsg, cbErrorMsgMax, *pcbErrorMsg);
```

for example, would be rewritten using the new function as:

```
SQLGetDiagRec(SQL_HANDLE_HSTMT, hstmt, 1, szSqlState, pfNativeError,
              szErrorMsg, cbErrorMsgMax, pcbErrorMsg);
```

Related concepts:
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Diagnostics in CLI applications overview" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:
- "SQLGetDiagField function (CLI) - Get a field of diagnostic data" on page 168
- "SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record" on page 173
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLExecDirect function (CLI) - Execute a statement directly

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLExecDirect()` directly executes the specified SQL statement or XQuery expression using the current values of the parameter marker variables if any parameters exist in the statement. The statement or expression can only be executed once.

For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the XMLQUERY function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in XMLQUERY for execution.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLExecDirectW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLExecDirect (
               SQLHSTMT          StatementHandle,         /* hstmt */
               SQLCHAR           *StatementText,          /* szSqlStr */
               SQLINTEGER        TextLength);             /* cbSqlStr */
```

Function arguments:

Table 45. SQLExecDirect arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. There must not be an open cursor associated with *StatementHandle*.
SQLCHAR *	*StatementText*	input	SQL statement or XQuery expression string.
SQLINTEGER	*TextLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *StatementText* argument, or SQL_NTS if *StatementText* is null-terminated.

Usage:

If the SQL statement text contains vendor escape clause sequences, DB2 CLI will first modify the SQL statement text to the appropriate DB2-specific format before submitting it for preparation and execution. If the application does not generate SQL statements that contain vendor escape clause sequences, then it should set the SQL_ATTR_NOSCAN statement attribute to SQL_NOSCAN_ON at the connection level so that DB2 CLI does not perform a scan for vendor escape clauses.

The SQL statement can be COMMIT or ROLLBACK if it is called using `SQLExecDirect()`. Doing so yields the same result as calling `SQLEndTran()` on the current connection handle.

The SQL statement string can contain parameter markers, however all parameters must be bound before calling SQLExecDirect().

If the SQL statement is a query, or *StatementText* is an XQuery expression, SQLExecDirect() will generate a cursor name, and open the cursor. If the application has used SQLSetCursorName() to associate a cursor name with the statement handle, DB2 CLI associates the application generated cursor name with the internally generated one.

If a result set is generated, SQLFetch() or SQLFetchScroll() will retrieve the next row (or rows) of data into bound variables, LOB locators, or LOB file references.

If the SQL statement is a positioned DELETE or a positioned UPDATE, the cursor referenced by the statement must be positioned on a row and must be defined on a separate statement handle under the same connection handle.

There must not already be an open cursor on the statement handle.

If SQLSetStmtAttr() has been called with the SQL_ATTR_PARAMSET_SIZE attribute to specify that an array of input parameter values has been bound to each parameter marker, then the application needs to call SQLExecDirect() only once to process the entire array of input parameter values.

If the executed statement returns multiple result sets (one for each set of input parameters), then SQLMoreResults() should be used to advance to the next result set once processing on the current result set is complete.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA
- SQL_NO_DATA_FOUND

SQL_NEED_DATA is returned when the application has requested to input data-at-execute parameter values by setting the **StrLen_or_IndPtr* value specified during SQLBindParameter() to SQL_DATA_AT_EXEC for one or more parameters.

SQL_NO_DATA_FOUND is returned if the SQL statement is a Searched UPDATE or Searched DELETE and no rows satisfy the search condition.

Diagnostics:

Table 46. SQLExecDirect SQLSTATEs

SQLSTATE	Description	Explanation
01504	The UPDATE or DELETE statement does not include a WHERE clause.	*StatementText* contained an UPDATE or DELETE statement which did not contain a WHERE clause. (Function returns SQL_SUCCESS_WITH_INFO or SQL_NO_DATA_FOUND if there were no rows in the table).
01508	Statement disqualified for blocking.	The statement was disqualified for blocking for reasons other than storage.

Table 46. SQLExecDirect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07001	Wrong number of parameters.	The number of parameters bound to application variables using `SQLBindParameter()` was less than the number of parameter markers in the SQL statement contained in the argument *StatementText*.
07006	Invalid conversion.	Transfer of data between DB2 CLI and the application variables would result in an incompatible data conversion.
21S01	Insert value list does not match column list.	*StatementText* contained an INSERT statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degrees of derived table does not match column list.	*StatementText* contained a CREATE VIEW statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22001	String data right truncation.	A character string assigned to a character type column exceeded the maximum length of the column.
22003	Numeric value out of range.	A numeric value assigned to a numeric type column caused truncation of the whole part of the number, either at the time of assignment or in computing an intermediate result. *StatementText* contained an SQL statement with an arithmetic expression which caused division by zero. **Note:** as a result the cursor state is undefined for DB2 Database for Linux, UNIX, and Windows (the cursor will remain open for other RDBMSs).
22005	Error in assignment.	*StatementText* contained an SQL statement with a parameter or literal and the value or LOB locator was incompatible with the data type of the associated table column. The length associated with a parameter value (the contents of the *pcbValue* buffer specified on `SQLBindParameter()`) is not valid. The argument *fSQLType* used in `SQLBindParameter()` or `SQLSetParam()`, denoted an SQL graphic data type, but the deferred length argument (*pcbValue*) contains an odd length value. The length value must be even for graphic data types.
22007	Invalid datetime format.	*StatementText* contained an SQL statement with an invalid datetime format; that is, an invalid string representation or value was specified, or the value was an invalid date, time, or timestamp.
22008	Datetime field overflow.	Datetime field overflow occurred; for example, an arithmetic operation on a date or timestamp has a result that is not within the valid range of dates, or a datetime value cannot be assigned to a bound variable because it is too small.
22012	Division by zero is invalid.	*StatementText* contained an SQL statement with an arithmetic expression that caused division by zero.
23000	Integrity constraint violation.	The execution of the SQL statement is not permitted because the execution would cause integrity constraint violation in the DBMS.
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
24504	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row.	Results were pending on the *StatementHandle* from a previous query or a cursor associated with the *hstmt* had not been closed.

Table 46. SQLExecDirect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
34000	Invalid cursor name.	*StatementText* contained a Positioned DELETE or a Positioned UPDATE and the cursor referenced by the statement being executed was not open.
37xxx [a]	Invalid SQL syntax.	*StatementText* contained one or more of the following: • an SQL statement that the connected database server could not prepare • a statement containing a syntax error
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42xxx	Syntax Error or Access Rule Violation.	**42**5xx indicates the authorization ID does not have permission to execute the SQL statement contained in *StatementText*. Other **42**xxx SQLSTATES indicate a variety of syntax or access problems with the statement.

428A1	Unable to access a file referenced by a host file variable.	This can be raised for any of the following scenarios. The associated reason code in the text identifies the particular error:
		• 01 - The file name length is invalid, or the file name, the path has an invalid format, or both.
		• 02 - The file option is invalid. It must have one of the following values:
		`SQL_FILE_READ -read from an existing file`
		`SQL_FILE_CREATE -create a new file for write`
		`SQL_FILE_OVERWRITE -overwrite an existing file. If the file does not exist, create the file.`
		`SQL_FILE_APPEND -append to an existing file. If the file does not exist, create the file.`
		• 03 - The file cannot be found.
		• 04 - The SQL_FILE_CREATE option was specified for a file with the same name as an existing file.
		• 05 - Access to the file was denied. The user does not have permission to open the file.
		• 06 - Access to the file was denied. The file is in use with incompatible modes. Files to be written to are opened in exclusive mode.
		• 07 - Disk full was encountered while writing to the file.
		• 08 - Unexpected end of file encountered while reading from the file.
		• 09 - A media error was encountered while accessing the file.

42895	The value of a host variable in the EXECUTE or OPEN statement cannot be used because of its data type.	The LOB locator type specified on the bind parameter function call does not match the LOB data type of the parameter marker. The argument *fSQLType* used on the bind parameter function specified a LOB locator type but the corresponding parameter marker is not a LOB.
44000	Integrity constraint violation.	*StatementText* contained an SQL statement which contained a parameter or literal. This parameter value was NULL for a column defined as NOT NULL in the associated table column, or a duplicate value was supplied for a column constrained to contain only unique values, or some other integrity constraint was violated.

Table 46. SQLExecDirect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
56084	LOB data is not supported in DRDA.	LOB columns cannot either be selected or updated when connecting to host or AS/400® servers (using DB2 Connect).
58004	Unexpected system failure.	Unrecoverable system error.
S0001	Database object already exists.	*StatementText* contained a CREATE TABLE or CREATE VIEW statement and the table name or view name specified already existed.
S0002	Database object does not exist.	*StatementText* contained an SQL statement that references a table name or view name which does not exist.
S0011	Index already exists.	*StatementText* contained a CREATE INDEX statement and the specified index name already existed.
S0012	Index not found.	*StatementText* contained a DROP INDEX statement and the specified index name did not exist.
S0021	Column already exists.	*StatementText* contained an ALTER TABLE statement and the column specified in the ADD clause was not unique or identified an existing column in the base table.
S0022	Column not found.	*StatementText* contained an SQL statement that references a column name which does not exist.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	*StatementText* was a null pointer.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The argument *TextLength* was less than 1 but not equal to SQL_NTS.
HY092	Option type out of range.	The *FileOptions* argument of a previous `SQLBindFileToParam()` operation was not valid.
HY503	Invalid file name length.	The *fileNameLength* argument value from `SQLBindFileToParam()` was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Note:

a xxx refers to any SQLSTATE with that class code. Example, **37**xxx refers to any SQLSTATE in the **37** class.

Restrictions:

None.

Example:

```
/* directly execute a statement - end the COMPOUND statement */
cliRC = SQLExecDirect(hstmt, (SQLCHAR *)"SELECT * FROM ORG", SQL_NTS);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

- "Parameter marker binding in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Vendor escape clauses in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Binding parameter markers in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Updating and deleting data in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "PREPARE statement" in *SQL Reference, Volume 2*
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLExecute function (CLI) - Execute a statement" on page 106
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "Statement attributes (CLI) list" on page 348

Related samples:

- "tbmod.c -- How to modify table data"
- "dbuse.c -- How to use a database"
- "dbmcon.c -- How to use multiple databases"

SQLExecute function (CLI) - Execute a statement

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLExecute()` executes a statement that was successfully prepared using `SQLPrepare()` on the same statement handle, once or multiple times. The statement is executed using the current values of any application variables that were bound to parameter markers by `SQLBindParameter()` or `SQLBindFileToParam()`.

Syntax:

```
SQLRETURN   SQLExecute (SQLHSTMT       StatementHandle);  /* hstmt */
```

Function arguments:

Table 47. SQLExecute arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. There must not be an open cursor associated with *StatementHandle*.

Usage:

The SQL statement string previously prepared on *StatementHandle* using SQLPrepare() may contain parameter markers. All parameters must be bound before calling `SQLExecute()`.

Note: For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the XMLQUERY function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in XMLQUERY for execution.

Once the application has processed the results from the `SQLExecute()` call, it can execute the statement again with new (or the same) parameter values.

A statement executed by `SQLExecDirect()` cannot be re-executed by calling `SQLExecute()`. Only statements prepared with `SQLPrepare()` can be executed and re-executed with `SQLExecute()`.

If the prepared SQL statement is a query or an XQuery expression, `SQLExecute()` will generate a cursor name, and open the cursor. If the application has used `SQLSetCursorName()` to associate a cursor name with the statement handle, DB2 CLI associates the application generated cursor name with the internally generated one.

To execute a query more than once on a given statement handle, the application must close the cursor by calling SQLCloseCursor() or `SQLFreeStmt()` with the SQL_CLOSE option. There must not be an open cursor on the statement handle when calling `SQLExecute()`.

If a result set is generated, `SQLFetch()` or `SQLFetchScroll()` will retrieve the next row (or rows) of data into bound variables, LOB locators or LOB file references.

If the SQL statement is a positioned DELETE or a positioned UPDATE, the cursor referenced by the statement must be positioned on a row at the time `SQLExecute()` is called, and must be defined on a separate statement handle under the same connection handle.

If `SQLSetStmtAttr()` has been called with the SQL_ATTR_PARAMSET_SIZE attribute to specify that an array of input parameter values has been bound to each parameter marker, the application needs to call `SQLExecute()` only once to process the entire array of input parameter values. If the executed statement returns multiple result sets (one for each set of input parameters), then `SQLMoreResults()` should be used to advance to the next result set once processing on the current result set is complete.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA
- SQL_NO_DATA_FOUND

SQL_NEED_DATA is returned when the application has requested to input data-at-execute parameter values by setting the **StrLen_or_IndPtr* value specified during `SQLBindParameter()` to SQL_DATA_AT_EXECUTE for one or more parameters.

SQL_NO_DATA_FOUND is returned if the SQL statement is a searched UPDATE or searched DELETE and no rows satisfy the search condition.

Diagnostics:

The SQLSTATEs for `SQLExecute()` include all those for `SQLExecDirect()` except for **HY**009, **HY**090 and with the addition of the SQLSTATE in the table below. Any SQLSTATE that `SQLPrepare()` could return can also be returned on a call to `SQLExecute()` as a result of deferred prepare behavior.

Table 48. SQLExecute SQLSTATEs

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The specified *StatementHandle* was not in a prepared state. `SQLExecute()` was called without first calling `SQLPrepare()`.

Authorization:

None.

Example:

```
SQLHANDLE hstmt; /* statement handle */
SQLCHAR *stmt = (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? ";
SQLSMALLINT parameter1 = 0;

/* allocate a statement handle */
cliRC = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

/* ... */

/* prepare the statement */
cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);

/* ... */

/* bind parameter1 to the statement */
cliRC = SQLBindParameter(hstmt,
                         1,
                         SQL_PARAM_INPUT,
                         SQL_C_SHORT,
                         SQL_SMALLINT,
                         0,
                         0,
                         &parameter1,
                         0,
                         NULL);
/* ... */
parameter1 = 15;

/* execute the statement for parameter1 = 15  */
cliRC = SQLExecute(hstmt);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Binding parameter markers in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Preparing and executing SQL statements in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Updating and deleting data in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter" on page 20
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLPrepare function (CLI) - Prepare a statement" on page 242
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spclient.c -- Call various stored procedures"
- "dbuse.c -- How to use a database"

SQLExtendedBind function (CLI) - Bind an array of columns

Purpose:

Specification:	DB2 CLI 6		

`SQLExtendedBind()` is used to bind an array of columns or parameters instead of using repeated calls to `SQLBindCol()` or `SQLBindParameter()`.

Syntax:

```
SQLRETURN    SQLExtendedBind (
                              SQLHSTMT          StatementHandle, /* hstmt */
                              SQLSMALLINT       fBindCol,
                              SQLSMALLINT       cRecords,
                              SQLSMALLINT *     pfCType,
                              SQLPOINTER  *     rgbValue,
                              SQLINTEGER  *     cbValueMax,
                              SQLUINTEGER *     puiPrecisionCType,
                              SQLSMALLINT *     psScaleCType,
                              SQLINTEGER  **    pcbValue,
                              SQLINTEGER  **    piIndicator,
                              SQLSMALLINT *     pfParamType,
                              SQLSMALLINT *     pfSQLType,
                              SQLUINTEGER *     pcbColDef,
                              SQLSMALLINT *     pibScale ) ;
```

Function arguments:

Table 49. SQLExtendedBind() arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.

Table 49. SQLExtendedBind() arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*fBindCol*	input	If SQL_TRUE then the result is similar to `SQLBindCol()`, otherwise, it is similar to `SQLBindParameter()`.
SQLSMALLINT	*cRecords*	input	Number of columns or parameters to bind.
SQLSMALLINT *	*pfCType*	input	Array of values for the application data type.
SQLPOINTER *	*rgbValue*	input	Array of pointers to application data area.
SQLINTEGER *	*cbValueMax*	input	Array of maximum sizes for *rgbValue*.
SQLUINTEGER *	*puiPrecisionCType*	input	Array of decimal precision values. Each value is used only if the application data type of the corresponding record is SQL_C_DECIMAL_IBM.
SQLSMALLINT *	*psScaleCType*	input	Array of decimal scale values. Each value is used only if the application data type of the corresponding record is SQL_C_DECIMAL_IBM.
SQLINTEGER **	*pcbValue*	input	Array of pointers to length values.
SQLINTEGER **	*piIndicator*	input	Array of pointers to indicator values.
SQLSMALLINT *	*pfParamType*	input	Array of parameter types. Only used if *fBindCol* is FALSE. Each row in this array serves the same purpose as the `SQLBindParameter()` argument *InputOutputType*. It can be set to: • SQL_PARAM_INPUT • SQL_PARAM_INPUT_OUTPUT • SQL_PARAM_OUTPUT
SQLSMALLINT *	*pfSQLType*	input	Array of SQL data types. Only used if *fBindCol* is FALSE. Each row in this array serves the same purpose as the `SQLBindParameter()` argument *ParameterType*.
SQLUINTEGER *	*pcbColDef*	input	Array of SQL precision values. Only used if *fBindCol* is FALSE. Each row in this array serves the same purpose as the `SQLBindParameter()` argument *ColumnSize*.
SQLSMALLINT *	*pibScale*	input	Array of SQL scale values. Only used if *fBindCol* is FALSE. Each row in this array serves the same purpose as the `SQLBindParameter()` argument *DecimalDigits*.

Usage:

The argument *fBindCol* determines whether this function call is used to associate (bind):

- parameter markers in an SQL statement (as with `SQLBindParameter()`) - *fBindCol* = SQL_FALSE
- columns in a result set (as with `SQLBindCol()`) - *fBindCol* = SQL_TRUE

This function can be used to replace multiple calls to `SQLBindCol()` or `SQLBindParameter()`, however, important differences should be noted. Depending on how the *fBindCol* parameter has been set, the input expected by `SQLExtendedBind()` is similar to either `SQLBindCol()` or `SQLBindParameter()` with the following exceptions:

- When SQLExtendedBind() is set to SQLBindCol() mode:
  - *targetValuePtr* must be a positive integer that specifies in bytes, the maximum length of the data that will be in the returned column.
- When SQLExtendedBind() is set to SQLBindParameter() mode:
  - *ColumnSize* must be a positive integer that specifies the maximum length of the target column in bytes, where applicable.
  - *DecimalDigits* must be set to the correct scale for the target column, where applicable.
  - *ValueType* of SQL_C_DEFAULT should not be used.
  - If *ValueType* is a locator type, the corresponding *ParameterType* should be a matching locator type.
  - All *ValueType* to *ParameterType* mappings should be as closely matched as possible to minimize the conversion that DB2 CLI must perform.

Each array reference passed to SQLExtendedBind() must contain at least the number of elements indicated by *cRecords*. If the calling application fails to pass in sufficiently large arrays, DB2 CLI may attempt to read beyond the end of the arrays resulting in corrupt data or critical application failure.

Each array passed to SQLExtendedBind() is considered to be a deferred argument, which means the values in the array are examined and retrieved at the time of execution. As a result, ensure that each array is in a valid state and contains valid data when DB2 CLI executes using the values in the array. Following a successful execution, if a statement needs to be executed again, you do not need to call SQLExtendedBind() a second time if the handles passed to the original call to SQLExtendedBind() still refer to valid arrays.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 50. SQLExtendedBind() SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The conversion from the data value identified by a row in the *pfCType* argument to the data type identified by the *pfParamType* argument is not a meaningful conversion. (For example, conversion from SQL_C_DATE to SQL_DOUBLE.)
07009	Invalid descriptor index	The value specified for the argument *cRecords* exceeded the maximum number of columns in the result set.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	A row in *pfParamType* or *pfSQLType* was not a valid data type or SQL_C_DEFAULT.

Table 50. SQLExtendedBind() SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY004	SQL data type out of range.	The value specified for the argument *pfParamType* is not a valid SQL data type.
HY009	Invalid argument value.	The argument *rgbValue* was a null pointer and the argument *cbValueMax* was a null pointer, and *pfParamType* is not SQL_PARAM_OUTPUT.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check was not consistent.
HY090	Invalid string or buffer length.	The value specified for the argument *cbValueMax* is less than 1 and the argument the corresponding row in *pfParamType* or *pfSQLType* is either SQL_C_CHAR, SQL_C_BINARY or SQL_C_DEFAULT.
HY093	Invalid parameter number.	The value specified for a row in the argument *pfCType* was less than 1 or greater than the maximum number of parameters supported by the server.
HY094	Invalid scale value.	The value specified for *pfParamType* was either SQL_DECIMAL or SQL_NUMERIC and the value specified for *DecimalDigits* was less than 0 or greater than the value for the argument *pcbColDef* (precision). The value specified for *pfParamType* was SQL_C_TIMESTAMP and the value for *pfParamType* was either SQL_CHAR or SQL_VARCHAR and the value for *DecimalDigits* was less than 0 or greater than 6.
HY104	Invalid precision value.	The value specified for *pfParamType* was either SQL_DECIMAL or SQL_NUMERIC and the value specified by *pcbColDef* was less than 1.
HY105	Invalid parameter type.	*pfParamType* is not one of SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT.
HYC00	Driver not capable.	DB2 CLI recognizes, but does not support the data type specified in the row in *pfParamType* or *pfSQLType*. A LOB locator C data type was specified, but the connected server does not support LOB data types.

Restrictions:

None.

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23

SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows)

Deprecated:

Note:

In ODBC 3.0, `SQLExtendedFetch()` has been deprecated and replaced with `SQLFetchScroll()`.

Although this version of DB2 CLI continues to support `SQLExtendedFetch()`, we recommend that you use `SQLFetchScroll()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLExtendedFetch(hstmt, SQL_FETCH_ABSOLUTE, 5, &rowCount, &rowStatus);
```

for example, would be rewritten using the new function as:

```
SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 5);
```

Note:

The information returned in the *rowCount* and *rowStatus* parameters of `SQLExtendedFetch()` are handled by `SQLFetchScroll()` as follows:

- *rowCount*: `SQLFetchScroll()` returns the number of rows fetched in the buffer pointed to by the SQL_ATTR_ROWS_FETCHED_PTR statement attribute.
- *rowStatus*: `SQLFetchScroll()` returns the array of statuses for each row in the buffer pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute.

Related reference:

- "CLI and ODBC function summary" on page 1
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "Statement attributes (CLI) list" on page 348

SQLExtendedPrepare function (CLI) - Prepare a statement and set statement attributes

Purpose:

Specification:	**DB2 CLI** 6.0		

SQLExtendedPrepare

SQLExtendedPrepare() is used to prepare a statement and set a group of statement attributes, all in one call.

This function can be used in place of a call to SQLPrepare() followed by a number of calls to SQLSetStmtAttr().

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLExtendedPrepareW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN SQLExtendedPrepare(
             SQLHSTMT      StatementHandle,  /* hstmt */
             SQLCHAR       *StatementText,   /* pszSqlStmt */
             SQLINTEGER    TextLength,       /* cbSqlStmt */
             SQLINTEGER    cPars,
             SQLSMALLINT   sStmtType,
             SQLINTEGER    cStmtAttrs,
             SQLINTEGER    *piStmtAttr,
             SQLINTEGER    *pvParams );
```

Function arguments:

Table 51. SQLExtendedPrepare() arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLCHAR *	*StatementText*	Input	SQL statement string.
SQLINTEGER	*TextLength*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *StatementText* argument, or SQL_NTS if *StatementText* is null-terminated.
SQLINTEGER	*cPars*	Input	Number of parameter markers in statement.
SQLSMALLINT	*cStmtType*	Input	Statement type. For possible values see "List of cStmtType Values" on page 115.
SQLINTEGER	*cStmtAttrs*	Input	Number of statement attributes specified on this call.
SQLINTEGER *	*piStmtAttr*	Input	Array of statement attributes to set.
SQLINTEGER *	*pvParams*	Input	Array of corresponding statement attributes values to set.

Usage:

The first three arguments of this function are exactly the same as the arguments in SQLPrepare().

There are two requirements when using SQLExtendedPrepare():

1. The SQL statements will not be scanned for ODBC/vendor escape clauses. It behaves as if the SQL_ATTR_NOSCAN statement attribute is set to SQL_NOSCAN. If the SQL statement contains ODBC/vendor escape clauses then SQLExtendedPrepare() cannot be used.
2. You must indicate in advance (through *cPars*) the number of parameter markers that are included in the SQL statement.

The *cPars* argument indicates the number of parameter markers in *StatementText*.

The argument *cStmtType* is used to indicate the type of statement that is being prepared. See "List of cStmtType Values" for the list of possible values.

The final three arguments are used to indicate a set of statement attributes to use. Set *cStmtAttrs* to the number of statement attributes specified on this call. Create two arrays, one to hold the list of statement attributes, one to hold the value for each. Use these arrays for *piStmtAttr* and *pvParams*.

List of cStmtType Values

The argument *cStmtType* can be set to one of the following values:
- SQL_CLI_STMT_UNDEFINED
- SQL_CLI_STMT_ALTER_TABLE
- SQL_CLI_STMT_CREATE_INDEX
- SQL_CLI_STMT_CREATE_TABLE
- SQL_CLI_STMT_CREATE_VIEW
- SQL_CLI_STMT_DELETE_SEARCHED
- SQL_CLI_STMT_DELETE_POSITIONED
- SQL_CLI_STMT_GRANT
- SQL_CLI_STMT_INSERT
- SQL_CLI_STMT_REVOKE
- SQL_CLI_STMT_SELECT
- SQL_CLI_STMT_UPDATE_SEARCHED
- SQL_CLI_STMT_UPDATE_POSITIONED
- SQL_CLI_STMT_CALL
- SQL_CLI_STMT_SELECT_FOR_UPDATE
- SQL_CLI_STMT_WITH
- SQL_CLI_STMT_SELECT_FOR_FETCH
- SQL_CLI_STMT_VALUES
- SQL_CLI_STMT_CREATE_TRIGGER
- SQL_CLI_STMT_SELECT_OPTIMIZE_FOR_NROWS
- SQL_CLI_STMT_SELECT_INTO
- SQL_CLI_STMT_CREATE_PROCEDURE
- SQL_CLI_STMT_CREATE_FUNCTION
- SQL_CLI_STMT_SET_CURRENT_QUERY_OPT

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 52. SQLExtendedPrepare SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01504	The UPDATE or DELETE statement does not include a WHERE clause.	*StatementText* contained an UPDATE or DELETE statement which did not contain a WHERE clause.
01508	Statement disqualified for blocking.	The statement was disqualified for blocking for reasons other than storage.

Table 52. SQLExtendedPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
01S02	Option value changed.	DB2 CLI did not support a value specified in **pvParams*, or a value specified in **pvParams* was invalid because of SQL constraints or requirements, so DB2 CLI substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
21S01	Insert value list does not match column list.	*StatementText* contained an INSERT statement and the number of values to be inserted did not match the degree of the derived table.
21S02	Degrees of derived table does not match column list.	*StatementText* contained a CREATE VIEW statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22018	Invalid character value for cast specification.	**StatementText* contained an SQL statement that contained a literal or parameter and the value was incompatible with the data type of the associated table column.
22019	Invalid escape character	The argument *StatementText* contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	The argument *StatementText* contained "LIKE *pattern value* ESCAPE *escape character*" in the WHERE clause, and the character following the escape character in the pattern value was not one of "%" or "_".
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
34000	Invalid cursor name.	*StatementText* contained a positioned DELETE or a positioned UPDATE and the cursor referenced by the statement being executed was not open.
37xxx [a]	Invalid SQL syntax.	*StatementText* contained one or more of the following: • an SQL statement that the connected database server could not prepare • a statement containing a syntax error
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to deadlock or timeout.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42xxx [a]	Syntax Error or Access Rule Violation.	**42**5xx indicates the authorization ID does not have permission to execute the SQL statement contained in *StatementText*. Other **42**xxx SQLSTATES indicate a variety of syntax or access problems with the statement.
58004	Unexpected system failure.	Unrecoverable system error.
S0001	Database object already exists.	*StatementText* contained a CREATE TABLE or CREATE VIEW statement and the table name or view name specified already existed.
S0002	Database object does not exist.	*StatementText* contained an SQL statement that references a table name or a view name which did not exist.
S0011	Index already exists.	*StatementText* contained a CREATE INDEX statement and the specified index name already existed.
S0012	Index not found.	*StatementText* contained a DROP INDEX statement and the specified index name did not exist.

Table 52. SQLExtendedPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
S0021	Column already exists.	*StatementText* contained an ALTER TABLE statement and the column specified in the ADD clause was not unique or identified an existing column in the base table.
S0022	Column not found.	*StatementText* contained an SQL statement that references a column name which did not exist.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	*StatementText* was a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY011	Operation invalid at this time.	The *Attribute* was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS and the statement was prepared.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY017	Invalid use of an automatically allocated descriptor handle.	The *Attribute* argument was SQL_ATTR_IMP_ROW_DESC or SQL_ATTR_IMP_PARAM_DESC. The *Attribute* argument was SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and the value in **ValuePtr* was an implicitly allocated descriptor handle.
HY024	Invalid attribute value.	Given the specified *Attribute* value, an invalid value was specified in **ValuePtr*. (DB2 CLI returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE. For all other connection and statement attributes, the driver must verify the value specified in **ValuePtr*.)
HY090	Invalid string or buffer length.	The argument *TextLength* was less than 1, but not equal to SQL_NTS.
HY092	Option type out of range.	The value specified for the argument *Attribute* was not valid for this version of DB2 CLI.
HYC00	Driver not capable.	The value specified for the argument *Attribute* was a valid connection or statement attribute for the version of the DB2 CLI driver, but was not supported by the data source.

Table 52. SQLExtendedPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Note:

a xxx refers to any SQLSTATE with that class code. Example, **37**xxx refers to any SQLSTATE in the **37** class.

Note: Not all DBMSs report all of the above diagnostic messages at prepare time. If deferred prepare is left on as the default behavior (controlled by the SQL_ATTR_DEFERRED_PREPARE statement attribute), then these errors could occur when the PREPARE is flowed to the server. The application must be able to handle these conditions when calling functions that cause this flow. These functions include `SQLExecute()`, `SQLDescribeParam()`, `SQLNumResultCols()`, `SQLDescribeCol()`, and `SQLColAttribute()`.

Restrictions:

None.

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLPrepare function (CLI) - Prepare a statement" on page 242
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "Statement attributes (CLI) list" on page 348
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLFetch function (CLI) - Fetch next row

Purpose:

Specification:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

`SQLFetch()` advances the cursor to the next row of the result set, and retrieves any bound columns.

Columns can be bound to:

- application storage
- LOB locators
- LOB file references

When `SQLFetch()` is called, the appropriate data transfer is performed, along with any data conversion if conversion was indicated when the column was bound. The columns can also be received individually after the fetch, by calling `SQLGetData()`.

`SQLFetch()` can only be called after a result set has been generated (using the same statement handle) by either executing a query, calling `SQLGetTypeInfo()` or calling a catalog function.

Syntax:

```
SQLRETURN   SQLFetch (SQLHSTMT      StatementHandle);  /* hstmt */
```

Function arguments:

Table 53. SQLFetch arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle

Usage:

`SQLFetch()` can only be called after a result set has been generated on the same statement handle. Before `SQLFetch()` is called the first time, the cursor is positioned before the start of the result set.

The number of application variables bound with `SQLBindCol()` must not exceed the number of columns in the result set or `SQLFetch()` will fail.

If `SQLBindCol()` has not been called to bind any columns, then `SQLFetch()` does not return data to the application, but just advances the cursor. In this case `SQLGetData()` could be called to obtain all of the columns individually. If the cursor is a multirow cursor (that is, the SQL_ATTR_ROW_ARRAY_SIZE is greater than 1), `SQLGetData()` can be called only if SQL_GD_BLOCK is returned when `SQLGetInfo()` is called with an *InfoType* of SQL_GETDATA_EXTENSIONS. (Not all DB2 data sources support SQL_GD_BLOCK.) Data in unbound columns is discarded when `SQLFetch()` advances the cursor to the next row. For fixed length data types, or small variable length data types, binding columns provides better performance than using `SQLGetData()`.

If LOB values are too large to be retrieved in one fetch, they can be retrieved in pieces by either using `SQLGetData()` (which can be used for any column type), or by binding a LOB locator, and using `SQLGetSubString()`.

If any bound storage buffer is not large enough to hold the data returned by `SQLFetch()`, the data will be truncated. If character data is truncated, SQL_SUCCESS_WITH_INFO is returned, and an SQLSTATE is generated indicating truncation. The `SQLBindCol()` deferred output argument *pcbValue* will contain the actual length of the column data retrieved from the server. The application should compare the actual output length to the input buffer length (*pcbValue* and *cbValueMax* arguments from `SQLBindCol()`) to determine which character columns have been truncated.

Truncation of numeric data types is reported as a warning if the truncation involves digits to the right of the decimal point. If truncation occurs to the left of the decimal point, an error is returned (refer to the diagnostics section).

Truncation of graphic data types is treated the same as character data types, except that the *rgbValue* buffer is filled to the nearest multiple of two bytes that is still less than or equal to the *cbValueMax* specified in `SQLBindCol()`. Graphic (DBCS) data transferred between DB2 CLI and the application is not null-terminated if the C buffer type is SQL_C_CHAR (unless the CLI/ODBC configuration keyword PATCH1 includes the value 64). If the buffer type is SQL_C_DBCHAR, then null-termination of graphic data does occur.

Truncation is also affected by the SQL_ATTR_MAX_LENGTH statement attribute. The application can specify that DB2 CLI should not report truncation by calling SQLSetStmtAttr() with SQL_ATTR_MAX_LENGTH and a value for the maximum length to return for any one column, and by allocating a *rgbValue* buffer of the same size (plus the null-terminator). If the column data is larger than the set maximum length, SQL_SUCCESS will be returned and the maximum length, not the actual length will be returned in *pcbValue*.

When all the rows have been retrieved from the result set, or the remaining rows are not needed, SQLCloseCursor() or SQLFreeStmt() with an option of SQL_CLOSE or SQL_DROP should be called to close the cursor and discard the remaining data and associated resources.

An application cannot mix SQLFetch() with SQLExtendedFetch() calls on the same statement handle. It can, however, mix SQLFetch() with SQLFetchScroll() calls on the same statement handle. Note that SQLExtendedFetch() has been deprecated and replaced with SQLFetchScroll().

Positioning the cursor

When the result set is created, the cursor is positioned before the start of the result set. SQLFetch() fetches the next rowset. It is equivalent to calling SQLFetchScroll() with *FetchOrientation* set to SQL_FETCH_NEXT.

The SQL_ATTR_ROW_ARRAY_SIZE statement attribute specifies the number of rows in the rowset. If the rowset being fetched by SQLFetch() overlaps the end of the result set, SQLFetch() returns a partial rowset. That is, if S + R-1 is greater than L, where S is the starting row of the rowset being fetched, R is the rowset size, and L is the last row in the result set, then only the first L-S+1 rows of the rowset are valid. The remaining rows are empty and have a status of SQL_ROW_NOROW.

Refer to the cursor positioning rules of SQL_FETCH_NEXT for SQLFetchScroll() for more information.

After SQLFetch() returns, the current row is the first row of the rowset.

Row status array

SQLFetch() sets values in the row status array in the same manner as SQLFetchScroll() and SQLBulkOperations(). The row status array is used to return the status of each row in the rowset. The address of this array is specified with the SQL_ATTR_ROW_STATUS_PTR statement attribute.

Rows fetched buffer

SQLFetch() returns the number of rows fetched in the rows fetched buffer including those rows for which no data was returned. The address of this buffer is specified with the SQL_ATTR_ROWSFETCHED_PTR statement attribute. The buffer is set by SQLFetch() and SQLFetchScroll().

Error handling

Errors and warnings can apply to individual rows or to the entire function. They can be retrieved using the SQLGetDiagField() function.

Errors and Warnings on the Entire Function

If an error applies to the entire function, such as SQLSTATE HYT00 (Timeout expired) or SQLSTATE 24000 (Invalid cursor state), SQLFetch() returns SQL_ERROR and the applicable SQLSTATE. The contents of the rowset buffers are undefined and the cursor position is unchanged.

If a warning applies to the entire function, SQLFetch() returns SQL_SUCCESS_WITH_INFO and the applicable SQLSTATE. The status records for warnings that apply to the entire function are returned before the status records that apply to individual rows.

Errors and warnings in individual rows

If an error (such as SQLSTATE 22012 (Division by zero)) or a warning (such as SQLSTATE 01004 (Data truncated)) applies to a single row, SQLFetch() returns SQL_SUCCESS_WITH_INFO, unless an error occurs in every row, in which case SQL_ERROR is returned. SQLFetch() also:

- Sets the corresponding element of the row status array to SQL_ROW_ERROR for errors or SQL_ROW_SUCCESS_WITH_INFO for warnings.
- Adds zero or more status records containing SQLSTATEs for the error or warning.
- Sets the row and column number fields in the status records. If SQLFetch() cannot determine a row or column number, it sets that number to SQL_ROW_NUMBER_UNKNOWN or SQL_COLUMN_NUMBER_UNKNOWN respectively. If the status record does not apply to a particular column, SQLFetch() sets the column number to SQL_NO_COLUMN_NUMBER.

SQLFetch() returns the status records in row number order. That is, it returns all status records for unknown rows (if any), then all status records for the first row (if any), then all status records for the second row (if any), and so on. The status records for each individual row are ordered according to the normal rules for ordering status records, described in SQLGetDiagField().

Descriptors and SQLFetch

The following sections describe how SQLFetch() interacts with descriptors.

Argument mappings

The driver does not set any descriptor fields based on the arguments of SQLFetch().

Other descriptor fields

The following descriptor fields are used by SQLFetch():

Table 54. Descriptor fields

Descriptor field	Desc.	Location	Set through
SQL_DESC_ARRAY_SIZE	ARD	header	SQL_ATTR_ROW_ARRAY_SIZE statement attribute
SQL_DESC_ARRAY_STATUS_PTR	IRD	header	SQL_ATTR_ROW_STATUS_PTR statement attribute
SQL_DESC_BIND_OFFSET_PTR	ARD	header	SQL_ATTR_ROW_BIND_OFFSET_PTR statement attribute

Table 54. Descriptor fields (continued)

Descriptor field	Desc.	Location	Set through
SQL_DESC_BIND_TYPE	ARD	header	SQL_ATTR_ROW_BIND_TYPE statement attribute
SQL_DESC_COUNT	ARD	header	*ColumnNumber* argument of `SQLBindCol()`
SQL_DESC_DATA_PTR	ARD	records	*TargetValuePtr* argument of `SQLBindCol()`
SQL_DESC_INDICATOR_PTR	ARD	records	*StrLen_or_IndPtr* argument in `SQLBindCol()`
SQL_DESC_OCTET_LENGTH	ARD	records	*BufferLength* argument in `SQLBindCol()`
SQL_DESC_OCTET_LENGTH_PTR	ARD	records	*StrLen_or_IndPtr* argument in `SQLBindCol()`
SQL_DESC_ROWS_PROCESSED_PTR	IRD	header	SQL_ATTR_ROWS_FETCHED_PTR statement attribute
SQL_DESC_TYPE	ARD	records	*TargetType* argument in `SQLBindCol()`

All descriptor fields can also be set through `SQLSetDescField()`.

Separate length and indicator buffers

Applications can bind a single buffer or two separate buffers to be used to hold length and indicator values. When an application calls `SQLBindCol()`, SQL_DESC_OCTET_LENGTH_PTR and SQL_DESC_INDICATOR_PTR fields of the ARD are set to the same address, which is passed in the *StrLen_or_IndPtr* argument. When an application calls `SQLSetDescField()` or `SQLSetDescRec()`, it can set these two fields to different addresses.

`SQLFetch()` determines whether the application has specified separate length and indicator buffers. In this case, when the data is not NULL, `SQLFetch()` sets the indicator buffer to 0 and returns the length in the length buffer. When the data is NULL, `SQLFetch()` sets the indicator buffer to SQL_NULL_DATA and does not modify the length buffer.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

SQL_NO_DATA_FOUND is returned if there are no rows in the result set, or previous `SQLFetch()` calls have fetched all the rows from the result set.

If all the rows have been fetched, the cursor is positioned after the end of the result set.

Diagnostics:

Table 55. SQLFetch SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data returned for one or more columns was truncated. String values or numeric values are right truncated. (SQL_SUCCESS_WITH_INFO is returned if no error occurred.)
07002	Too many columns.	A column number specified in the binding for one or more columns was greater than the number of columns in the result set.

Table 55. SQLFetch SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The data value could not be converted in a meaningful manner to the data type specified by *fCType* in `SQLBindCol()`
07009	Invalid descriptor index	Column 0 was bound but bookmarks are not being used (the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF).
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument *pcbValue* in `SQLBindCol()` was a null pointer and the value of the corresponding column is null. There is no means to report SQL_NULL_DATA. The pointer specified for the argument `IndicatorValue` in `SQLBindFileToCol()` was a null pointer and the value of the corresponding LOB column is NULL. There is no means to report SQL_NULL_DATA.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for one or more columns would have caused the whole part of the number to be truncated either at the time of assignment or in computing an intermediate result. A value from an arithmetic expression was returned which resulted in division by zero. **Note:** The associated cursor is undefined if this error is detected by DB2 Database for Linux, UNIX, and Windows. If the error was detected by DB2 CLI or by other IBM RDBMSs, the cursor will remain open and continue to advance on subsequent fetch calls.
22005	Error in assignment.	A returned value was incompatible with the data type of binding. A returned LOB locator was incompatible with the data type of the bound column.
22007	Invalid datetime format.	Conversion from character a string to a datetime format was indicated, but an invalid string representation or value was specified, or the value was an invalid date. The value of a date, time, or timestamp does not conform to the syntax for the specified data type.
22008	Datetime field overflow.	Datetime field overflow occurred; for example, an arithmetic operation on a date or timestamp has a result that is not within the valid range of dates, or a datetime value cannot be assigned to a bound variable because it is too small.
22012	Division by zero is invalid.	A value from an arithmetic expression was returned which resulted in division by zero.
24000	Invalid cursor state.	The previous SQL statement executed on the statement handle was not a query.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.

Table 55. SQLFetch SQLSTATEs (continued)

SQLSTATE	Description	Explanation

428A1	Unable to access a file referenced by a host file variable.	This can be raised for any of the following scenarios. The associated reason code in the text identifies the particular error:
		• 01 - The file name length is invalid, or the file name, the path or both has an invalid format.
		• 02 - The file option is invalid. It must have one of the following values:
		`SQL_FILE_READ -read from an existing file`
		`SQL_FILE_CREATE -create a new file for write`
		`SQL_FILE_OVERWRITE -overwrite an existing file. If the file does not exist, create the file.`
		`SQL_FILE_APPEND -append to an existing file. If the file does not exist, create the file.`
		• 03 - The file cannot be found.
		• 04 - The SQL_FILE_CREATE option was specified for a file with the same name as an existing file.
		• 05 - Access to the file was denied. The user does not have permission to open the file.
		• 06 - Access to the file was denied. The file is in use with incompatible modes. Files to be written to are opened in exclusive mode.
		• 07 - Disk full was encountered while writing to the file.
		• 08 - Unexpected end of file encountered while reading from the file.
		• 09 - A media error was encountered while accessing the file.

54028	The maximum number of concurrent LOB handles has been reached.	Maximum LOB locator assigned. The maximum number of concurrent LOB locators has been reached. A new locator can not be assigned.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	`SQLFetch()` was called for an StatementHandle after `SQLExtendedFetch()` was called and before `SQLFreeStmt()` had been called with the SQL_CLOSE option. The function was called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Table 55. SQLFetch SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY092	Option type out of range.	The *FileOptions* argument of a previous `SQLBindFileToCol()` operation was not valid.
HYC00	Driver not capable.	DB2 CLI or the data source does not support the conversion specified by the combination of the *fCType* in `SQLBindCol()` or `SQLBindFileToCol()` and the SQL data type of the corresponding column. A call to `SQLBindCol()` was made for a column data type which is not supported by DB2 CLI.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* fetch each row and display */
cliRC = SQLFetch(hstmt);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

if (cliRC == SQL_NO_DATA_FOUND)
{
  printf("\n  Data not found.\n");
}
while (cliRC != SQL_NO_DATA_FOUND)
{
  printf("    %-8d %-14.14s \n", deptnumb.val, location.val);

  /* fetch next row */
  cliRC = SQLFetch(hstmt);
  STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
}
```

Related concepts:

- "Result set terminology in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Retrieving query results in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Cursor positioning rules for SQLFetchScroll() (CLI)" on page 132
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column" on page 16
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLExecute function (CLI) - Execute a statement" on page 106

- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "SQLGetData function (CLI) - Get data from a column" on page 152
- "SQLGetDiagField function (CLI) - Get a field of diagnostic data" on page 168
- "Statement attributes (CLI) list" on page 348
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbread.c -- How to read data from tables"
- "dbuse.c -- How to use a database"

SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

`SQLFetchScroll()` fetches the specified rowset of data from the result set and returns data for all bound columns. Rowsets can be specified at an absolute or relative position or by bookmark.

Syntax:

```
SQLRETURN   SQLFetchScroll   (SQLHSTMT          StatementHandle,
                              SQLSMALLINT       FetchOrientation,
                              SQLLEN            FetchOffset);
```

Function arguments:

Table 56. SQLFetchScroll arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLUSMALLINT	*FetchOrientation*	input	Type of fetch: • SQL_FETCH_NEXT • SQL_FETCH_PRIOR • SQL_FETCH_FIRST • SQL_FETCH_LAST • SQL_FETCH_ABSOLUTE • SQL_FETCH_RELATIVE • SQL_FETCH_BOOKMARK For more information, see "Positioning the Cursor" on page 127.
SQLLEN	*FetchOffset*	input	Number of the row to fetch. The interpretation of this argument depends on the value of the *FetchOrientation* argument. For more information, see "Positioning the Cursor" on page 127.

Usage:

Overview

SQLFetchScroll() returns a specified rowset from the result set. Rowsets can be specified by absolute or relative position or by bookmark. SQLFetchScroll() can be called only while a result set exists, that is, after a call that creates a result set and before the cursor over that result set is closed. If any columns are bound, it returns the data in those columns. If the application has specified a pointer to a row status array or a buffer in which to return the number of rows fetched, SQLFetchScroll() returns this information as well. Calls to SQLFetchScroll() can be mixed with calls to SQLFetch() but cannot be mixed with calls to SQLExtendedFetch().

Positioning the cursor

When the result set is created, the cursor is positioned before the start of the result set. SQLFetchScroll() positions the block cursor based on the values of the *FetchOrientation* and *FetchOffset* arguments as shown in the following table. The exact rules for determining the start of the new rowset are shown in the next section.

FetchOrientation	Meaning
SQL_FETCH_NEXT	Return the next rowset. This is equivalent to calling SQLFetch(). SQLFetchScroll() ignores the value of *FetchOffset*.
SQL_FETCH_PRIOR	Return the prior rowset. SQLFetchScroll() ignores the value of *FetchOffset*.
SQL_FETCH_RELATIVE	Return the rowset *FetchOffset* from the start of the current rowset.
SQL_FETCH_ABSOLUTE	Return the rowset starting at row *FetchOffset*.
SQL_FETCH_FIRST	Return the first rowset in the result set. SQLFetchScroll() ignores the value of *FetchOffset*.
SQL_FETCH_LAST	Return the last complete rowset in the result set. SQLFetchScroll() ignores the value of *FetchOffset*.
SQL_FETCH_BOOKMARK	Return the rowset *FetchOffset* rows from the bookmark specified by the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute.

Not all cursors support all of these options. A static forward-only cursor, for example, will only support SQL_FETCH_NEXT. Scrollable cursors, such as keyset cursors, will support all of these options. The SQL_ATTR_ROW_ARRAY_SIZE statement attribute specifies the number of rows in the rowset. If the rowset being fetched by SQLFetchScroll() overlaps the end of the result set, SQLFetchScroll() returns a partial rowset. That is, if S + R-1 is greater than L, where S is the starting row of the rowset being fetched, R is the rowset size, and L is the last row in the result set, then only the first L-S+1 rows of the rowset are valid. The remaining rows are empty and have a status of SQL_ROW_NOROW.

After SQLFetchScroll() returns, the rowset cursor is positioned on the first row of the result set.

Returning data in bound columns

SQLFetchScroll() returns data in bound columns in the same way as SQLFetch().

If no columns are bound, SQLFetchScroll() does not return data but does move the block cursor to the specified position. As with SQLFetch(), you can use SQLGetData() to retrieve the information in this case.

Row status array

The row status array is used to return the status of each row in the rowset. The address of this array is specified with the SQL_ATTR_ROW_STATUS_PTR statement attribute. The array is allocated by the application and must have as many elements as are specified by the SQL_ATTR_ROW_ARRAY_SIZE statement attribute. Its values are set by SQLFetch(), SQLFetchScroll(), or SQLSetPos() (except when they have been called after the cursor has been positioned by SQLExtendedFetch()). If the value of the SQL_ATTR_ROW_STATUS_PTR statement attribute is a null pointer, these functions do not return the row status.

The contents of the row status array buffer are undefined if SQLFetch() or SQLFetchScroll() does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO.

The following values are returned in the row status array.

Row status array value	Description
SQL_ROW_SUCCESS	The row was successfully fetched.
SQL_ROW_SUCCESS_WITH_INFO	The row was successfully fetched. However, a warning was returned about the row.
SQL_ROW_ERROR	An error occurred while fetching the row.
SQL_ROW_ADDED	The row was inserted by SQLBulkOperations(). If the row is fetched again, or is refreshed by SQLSetPos() its status is SQL_ROW_SUCCESS. This value is not set by SQLFetch() or SQLFetchScroll().
SQL_ROW_UPDATED	The row was successfully fetched and has changed since it was last fetched from this result set. If the row is fetched again from this result set, or is refreshed by SQLSetPos(), the status changes to the row's new status.
SQL_ROW_DELETED	The row has been deleted since it was last fetched from this result set.
SQL_ROW_NOROW	The rowset overlapped the end of the result set and no row was returned that corresponded to this element of the row status array.

Rows fetched buffer

The rows fetched buffer is used to return the number of rows fetched, including those rows for which no data was returned because an error occurred while they were being fetched. In other words, it is the number of rows for which the value in the row status array is not SQL_ROW_NOROW. The address of this buffer is specified with the SQL_ATTR_ROWS_FETCHED_PTR statement attribute. The buffer is allocated by the application. It is set by SQLFetch() and SQLFetchScroll(). If the value of the SQL_ATTR_ROWS_FETCHED_PTR statement attribute is a null pointer, these functions do not return the number of rows fetched. To determine

the number of the current row in the result set, an application can call SQLGetStmtAttr() with the SQL_ATTR_ROW_NUMBER attribute.

The contents of the rows fetched buffer are undefined if SQLFetch() or SQLFetchScroll() does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, except when SQL_NO_DATA is returned, in which case the value in the rows fetched buffer is set to 0.

Error handling

SQLFetchScroll() returns errors and warnings in the same manner as SQLFetch().

Descriptors and SQLFetchScroll()

SQLFetchScroll() interacts with descriptors in the same manner as SQLFetch().

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

The return code associated with each SQLSTATE value is SQL_ERROR, unless noted otherwise. If an error occurs on a single column, SQLGetDiagField() can be called with a *DiagIdentifier* of SQL_DIAG_COLUMN_NUMBER to determine the column the error occurred on; and SQLGetDiagField() can be called with a *DiagIdentifier* of SQL_DIAG_ROW_NUMBER to determine the row containing that column.

Table 57. SQLFetchScroll SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	String or binary data returned for a column resulted in the truncation of non-blank character or non-NULL binary data. String values are right truncated. (Function returns SQL_SUCCESS_WITH_INFO.)
01S01	Error in row.	An error occurred while fetching one or more rows. (Function returns SQL_SUCCESS_WITH_INFO.) (This SQLSTATE is only returned when connected to DB2 CLI v2.)
01S06	Attempt to fetch before the result set returned the first rowset.	The requested rowset overlapped the start of the result set when the current position was beyond the first row, and either *FetchOrientation* was SQL_PRIOR, or *FetchOrientation* was SQL_RELATIVE with a negative *FetchOffset* whose absolute value was less than or equal to the current SQL_ATTR_ROW_ARRAY_SIZE. (Function returns SQL_SUCCESS_WITH_INFO.)
01S07	Fractional truncation.	The data returned for a column was truncated. For numeric data types, the fractional part of the number was truncated. For time or timestamp data types, the fractional portion of the time was truncated.

Table 57. SQLFetchScroll SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07002	Too many columns.	A column number specified in the binding for one or more columns was greater than the number of columns in the result set.
07006	Invalid conversion.	A data value of a column in the result set could not be converted to the C data type specified by *TargetType* in `SQLBindCol()`.
07009	Invalid descriptor index.	Column 0 was bound and the SQL_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
22001	String data right truncation.	A variable-length bookmark returned for a row was truncated.
22002	Invalid output or indicator buffer specified.	NULL data was fetched into a column whose *StrLen_or_IndPtr* set by `SQLBindCol()` (or SQL_DESC_INDICATOR_PTR set by `SQLSetDescField()` or `SQLSetDescRec()`) was a null pointer.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for one or more bound columns would have caused the whole (as opposed to fractional) part of the number to be truncated.
22007	Invalid datetime format.	A character column in the result set was bound to a date, time, or timestamp C structure, and a value in the column was, respectively, an invalid date, time, or timestamp.
22012	Division by zero is invalid.	A value from an arithmetic expression was returned which resulted in division by zero.
22018	Invalid character value for cast specification.	A character column in the result set was bound to a character C buffer and the column contained a character for which there was no representation in the character set of the buffer. A character column in the result set was bound to an approximate numeric C buffer and a value in the column could not be cast to a valid approximate numeric value. A character column in the result set was bound to an exact numeric C buffer and a value in the column could not be cast to a valid exact numeric value. A character column in the result set was bound to a datetime C buffer and a value in the column could not be cast to a valid datetime value.
24000	Invalid cursor state.	The *StatementHandle* was in an executed state but no result set was associated with the *StatementHandle*.
40001	Transaction rollback.	The transaction in which the fetch was executed was terminated to prevent deadlock.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.

Table 57. SQLFetchScroll SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The specified *StatementHandle* was not in an executed state. The function was called without first calling `SQLExecDirect()`, `SQLExecute()`, or a catalog function. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()` or `SQLExecDirect()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. `SQLFetchScroll()` was called for a *StatementHandle* after `SQLExtendedFetch()` was called and before `SQLFreeStmt()` with SQL_CLOSE was called.
HY106	Fetch type out of range.	The value specified for the argument *FetchOrientation* was invalid. The argument *FetchOrientation* was SQL_FETCH_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF. The value of the SQL_CURSOR_TYPE statement attribute was SQL_CURSOR_FORWARD_ONLY and the value of argument *FetchOrientation* was not SQL_FETCH_NEXT.
HY107	Row value out of range.	The value specified with the SQL_ATTR_CURSOR_TYPE statement attribute was SQL_CURSOR_KEYSET_DRIVEN, but the value specified with the SQL_ATTR_KEYSET_SIZE statement attribute was greater than 0 and less than the value specified with the SQL_ATTR_ROW_ARRAY_SIZE statement attribute.
HY111	Invalid bookmark value.	The argument *FetchOrientation* was SQL_FETCH_BOOKMARK and the bookmark pointed to by the value in the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute was not valid or was a null pointer.
HYC00	Driver not capable.	The specified fetch type is not supported. The conversion specified by the combination of the *TargetType* in `SQLBindCol()` and the SQL data type of the corresponding column is not supported.

Restrictions:

None.

Example:

```
/* fetch the rowset: row15, row16, row17, row18, row19 */
printf("\n  Fetch the rowset: row15, row16, row17, row18, row19.\n");

/* fetch the rowset and return data for all bound columns */
cliRC = SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 15);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* call SQLFetchScroll with SQL_FETCH_RELATIVE offset 3 */
printf("  SQLFetchScroll with SQL_FETCH_RELATIVE offset 3.\n");
printf("    COL1          COL2         \n");
printf("    ------------  -------------\n");
```

```
/* fetch the rowset and return data for all bound columns */
cliRC = SQLFetchScroll(hstmt, SQL_FETCH_RELATIVE, 3);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Diagnostics in CLI applications overview" in *Call Level Interface Guide and Reference, Volume 1*
- "Result set terminology in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Cursor positioning rules for SQLFetchScroll() (CLI)" on page 132
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLExtendedFetch function (CLI) - Extended fetch (fetch array of rows)" on page 113
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLGetDiagField function (CLI) - Get a field of diagnostic data" on page 168
- "SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute" on page 216
- "SQLSetPos function (CLI) - Set the cursor position in a rowset" on page 287
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbread.c -- How to read data from tables"

Cursor positioning rules for SQLFetchScroll() (CLI)

The following sections describe the exact rules for each value of *FetchOrientation*. These rules use the following notation:

FetchOrientation	Meaning
Before start	The block cursor is positioned before the start of the result set. If the first row of the new rowset is before the start of the result set, `SQLFetchScroll()` returns SQL_NO_DATA.
After end	The block cursor is positioned after the end of the result set. If the first row of the new rowset is after the end of the result set, `SQLFetchScroll()` returns SQL_NO_DATA.
CurrRowsetStart	The number of the first row in the current rowset.
LastResultRow	The number of the last row in the result set.
RowsetSize	The rowset size.
FetchOffset	The value of the *FetchOffset* argument.

BookmarkRow

The row corresponding to the bookmark specified by the SQL_ATTR_FETCH_BOOKMARK_PTR statement attribute.

SQL_FETCH_NEXT rules:

Table 58. SQL_FETCH_NEXT rules:

Condition	First row of new rowset
Before start	1
CurrRowsetStart + RowsetSize <= LastResultRow	CurrRowsetStart + RowsetSize
CurrRowsetStart + RowsetSize > LastResultRow	After end
After end	After end

SQL_FETCH_PRIOR rules:

Table 59. SQL_FETCH_PRIOR rules:

Condition	First row of new rowset
Before start	Before start
CurrRowsetStart = 1	Before start
1 < CurrRowsetStart <= RowsetSize	1 [a]
CurrRowsetStart > RowsetSize	CurrRowsetStart - RowsetSize
After end AND LastResultRow < RowsetSize	1 [a]
After end AND LastResultRow >= RowsetSize	LastResultRow - RowsetSize + 1

a `SQLFetchScroll()` returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset.) and SQL_SUCCESS_WITH_INFO.

SQL_FETCH_RELATIVE rules:

Table 60. SQL_FETCH_RELATIVE rules:

Condition	First row of new rowset
(Before start AND *FetchOffset* > 0) OR (After end AND *FetchOffset* < 0)	-- [a]
Before start AND *FetchOffset* <= 0	Before start
CurrRowsetStart = 1 AND FetchOffset < 0	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| > RowsetSize	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + *FetchOffset* < 1 AND |*FetchOffset*| <= RowsetSize	1 [b]
1 <= CurrRowsetStart + *FetchOffset* <= LastResultRow	CurrRowsetStart + *FetchOffset*
CurrRowsetStart + *FetchOffset* > LastResultRow	After end
After end AND *FetchOffset* >= 0	After end

a `SQLFetchScroll()` returns the same rowset as if it was called with *FetchOrientation* set to SQL_FETCH_ABSOLUTE. For more information, see the "SQL_FETCH_ABSOLUTE" section.
b `SQLFetchScroll()` returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset.) and SQL_SUCCESS_WITH_INFO.

SQL_FETCH_ABSOLUTE rules:

Table 61. SQL_FETCH_ABSOLUTE rules:

Condition	First row of new rowset
FetchOffset < 0 AND |*FetchOffset*| <= LastResultRow	LastResultRow + *FetchOffset* + 1
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |FetchOffset| > RowsetSize	Before start
FetchOffset < 0 AND |*FetchOffset*| > LastResultRow AND |FetchOffset| <= RowsetSize	1 [a]
FetchOffset = 0	Before start
1 <= *FetchOffset* <= LastResultRow	*FetchOffset*
FetchOffset > LastResultRow	After end

a `SQLFetchScroll()` returns SQLSTATE 01S06 (Attempt to fetch before the result set returned the first rowset.) and SQL_SUCCESS_WITH_INFO.

SQL_FETCH_FIRST rules:

Table 62. SQL_FETCH_FIRST rules:

Condition	First row of new rowset
Any	1

SQL_FETCH_LAST rules:

Table 63. SQL_FETCH_LAST rules:

Condition	First row of new rowset
RowsetSize <= LastResultRow	LastResultRow - RowsetSize + 1
RowsetSize > LastResultRow	1

SQL_FETCH_BOOKMARK rules:

Table 64. SQL_FETCH_BOOKMARK rules:

Condition	First row of new rowset
BookmarkRow + *FetchOffset* < 1	Before start
1 <= BookmarkRow + *FetchOffset* <= LastResultRow	BookmarkRow +*FetchOffset*
BookmarkRow + *FetchOffset* > LastResultRow	After end

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126

SQLForeignKeys function (CLI) - Get the list of foreign key columns

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

SQLForeignKeys() returns information about foreign keys for the specified table. The information is returned in an SQL result set which can be processed using the same functions that are used to retrieve a result generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLForeignKeysW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLForeignKeys   (
                SQLHSTMT         StatementHandle,  /* hstmt */
                SQLCHAR          *PKCatalogName,   /* szPkCatalogName */
                SQLSMALLINT      NameLength1,      /* cbPkCatalogName */
                SQLCHAR          *PKSchemaName,    /* szPkSchemaName */
                SQLSMALLINT      NameLength2,      /* cbPkSchemaName */
                SQLCHAR          *PKTableName,     /* szPkTableName */
                SQLSMALLINT      NameLength3,      /* cbPkTableName */
                SQLCHAR          *FKCatalogName,   /* szFkCatalogName */
                SQLSMALLINT      NameLength4,      /* cbFkCatalogName */
                SQLCHAR          *FKSchemaName,    /* szFkSchemaName */
                SQLSMALLINT      NameLength5,      /* cbFkSchemaName */
                SQLCHAR          *FKTableName,     /* szFkTableName */
                SQLSMALLINT      NameLength6);     /* cbFkTableName */
```

Function arguments:

Table 65. SQLForeignKeys arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR *	*PKCatalogName*	input	Catalog qualifier of the 3-part primary key table name. If the target DBMS does not support 3-part naming, and *PKCatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *PKCatalogName*, or SQL_NTS if *PKCatalogName* is null-terminated.
SQLCHAR *	*PKSchemaName*	input	Schema qualifier of the primary key table.
SQLSMALLINT	*NameLength2*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *PKSchemaName*, or SQL_NTS if *PKSchemaName* is null-terminated.
SQLCHAR *	*PKTableName*	input	Name of the table name containing the primary key.
SQLSMALLINT	*NameLength3*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *PKTableName*, or SQL_NTS if *PKTableName* is null-terminated.

Table 65. SQLForeignKeys arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	*FKCatalogName*	input	Catalog qualifier of the 3-part foreign key table name. If the target DBMS does not support 3-part naming, and *FKCatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength4*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *FKCatalogName*, or SQL_NTS if *FKCatalogName* is null-terminated.
SQLCHAR *	*FKSchemaName*	input	Schema qualifier of the table containing the foreign key.
SQLSMALLINT	*NameLength5*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *FKSchemaName*, or SQL_NTS if *FKSchemaName* is null-terminated.
SQLCHAR *	*FKTableName*	input	Name of the table containing the foreign key.
SQLSMALLINT	*NameLength6*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *FKTableName*, or SQL_NTS if *FKTableName* is null-terminated.

Usage:

If *PKTableName* contains a table name, and *FKTableName* is an empty string, `SQLForeignKeys()` returns a result set containing the primary key of the specified table and all of the foreign keys (in other tables) that refer to it.

If *FKTableName* contains a table name, and *PKTableName* is an empty string, `SQLForeignKeys()` returns a result set containing all of the foreign keys in the specified table and the primary keys (in other tables) to which they refer.

If both *PKTableName* and *FKTableName* contain table names, `SQLForeignKeys()` returns the foreign keys in the table specified in *FKTableName* that refer to the primary key of the table specified in *PKTableName*. This should be one key at the most.

If the schema qualifier argument associated with a table name is not specified, then the schema name defaults to the one currently in effect for the current connection.

"Columns Returned by SQLForeignKeys" on page 137 lists the columns of the result set generated by the `SQLForeignKeys()` call. If the foreign keys associated with a primary key are requested, the result set is ordered by FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, and ORDINAL_POSITION. If the primary keys associated with a foreign key are requested, the result set is ordered by PKTABLE_CAT, PKTABLE_SCHEM, PKTABLE_NAME, and ORDINAL_POSITION.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room

for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the associated TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns returned by SQLForeignKeys

Column 1 PKTABLE_CAT (VARCHAR(128))
Name of the catalog for PKTABLE_NAME. The value is NULL if this table does not have catalogs.

Column 2 PKTABLE_SCHEM (VARCHAR(128))
Name of the schema containing PKTABLE_NAME.

Column 3 PKTABLE_NAME (VARCHAR(128) not NULL)
Name of the table containing the primary key.

Column 4 PKCOLUMN_NAME (VARCHAR(128) not NULL)
Primary key column name.

Column 5 FKTABLE_CAT (VARCHAR(128))
Name of the catalog for FKTABLE_NAME. The value is NULL if this table does not have catalogs.

Column 6 FKTABLE_SCHEM (VARCHAR(128))
Name of the schema containing FKTABLE_NAME.

Column 7 FKTABLE_NAME (VARCHAR(128) not NULL)
Name of the table containing the foreign key.

Column 8 FKCOLUMN_NAME (VARCHAR(128) not NULL)
Foreign key column name.

Column 9 KEY_SEQ (SMALLINT not NULL)
Ordinal position of the column in the key, starting at 1.

Column 10 UPDATE_RULE (SMALLINT)
Action to be applied to the foreign key when the SQL operation is UPDATE:
- SQL_RESTRICT
- SQL_NO_ACTION

The update rule for IBM DB2 DBMSs is always either RESTRICT or SQL_NO_ACTION. However, ODBC applications might encounter the following UPDATE_RULE values when connected to non-IBM RDBMSs:
- SQL_CASCADE
- SQL_SET_NULL

Column 11 DELETE_RULE (SMALLINT)
Action to be applied to the foreign key when the SQL operation is DELETE:
- SQL_CASCADE
- SQL_NO_ACTION
- SQL_RESTRICT
- SQL_SET_DEFAULT
- SQL_SET_NULL

Column 12 FK_NAME (VARCHAR(128))
Foreign key identifier. NULL if not applicable to the data source.

Column 13 PK_NAME (VARCHAR(128))
Primary key identifier. NULL if not applicable to the data source.

Column 14 DEFERRABILITY (SMALLINT)
One of:
- SQL_INITIALLY_DEFERRED
- SQL_INITIALLY_IMMEDIATE
- SQL_NOT_DEFERRABLE

Note: The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents and order are identical to those defined for the `SQLForeignKeys()` result set in ODBC.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 66. SQLForeignKeys SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor is already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	The arguments *PKTableName* and *FKTableName* were both NULL pointers.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called For the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The length of the table or owner name is greater than the maximum length supported by the server.

Table 66. SQLForeignKeys SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get the list of foreign key columns */
cliRC = SQLForeignKeys(hstmt,
                       NULL,
                       0,
                       tbSchema,
                       SQL_NTS,
                       tbName,
                       SQL_NTS,
                       NULL,
                       0,
                       NULL,
                       SQL_NTS,
                       NULL,
                       SQL_NTS);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Foreign keys in a referential constraint" in *Administration Guide: Implementation*

Related reference:

- "SQLPrimaryKeys function (CLI) - Get primary key columns of a table" on page 247
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbconstr.c -- How to work with constraints associated with tables"

SQLFreeConnect function (CLI) - Free connection handle

Deprecated:

Note:

In ODBC 3.0, `SQLFreeConnect()` has been deprecated and replaced with `SQLFreeHandle()`.

Although this version of DB2 CLI continues to support SQLFreeConnect(), we recommend that you use SQLFreeHandle() in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLFreeConnect(hdbc);
```

for example, would be rewritten using the new function as:

```
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
```

Related reference:

- "SQLDisconnect function (CLI) - Disconnect from a data source" on page 89
- "SQLFreeHandle function (CLI) - Free handle resources" on page 140

SQLFreeEnv function (CLI) - Free environment handle

Deprecated:

Note:

In ODBC 3.0, SQLFreeEnv() has been deprecated and replaced with SQLFreeHandle().

Although this version of DB2 CLI continues to support SQLFreeEnv(), we recommend that you use SQLFreeHandle() in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLFreeEnv(henv);
```

for example, would be rewritten using the new function as:

```
SQLFreeHandle(SQL_HANDLE_ENV, henv);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLFreeHandle function (CLI) - Free handle resources" on page 140

SQLFreeHandle function (CLI) - Free handle resources

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLFreeHandle() frees resources associated with a specific environment, connection, statement, or descriptor handle.

Note: This function is a generic function for freeing resources. It replaces the ODBC 2.0 functions `SQLFreeConnect()` (for freeing a connection handle), and `SQLFreeEnv()` (for freeing an environment handle). `SQLFreeHandle()` also replaces the ODBC 2.0 function `SQLFreeStmt()` (with the SQL_DROP Option) for freeing a statement handle.

Syntax:

```
SQLRETURN   SQLFreeHandle (
               SQLSMALLINT       HandleType,  /* fHandleType */
               SQLHANDLE         Handle);     /* hHandle */
```

Function arguments:

Table 67. SQLFreeHandle arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	The type of handle to be freed by `SQLFreeHandle()`. Must be one of the following values: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC If *HandleType* is not one of the above values, `SQLFreeHandle()` returns SQL_INVALID_HANDLE.
SQLHANDLE	*Handle*	input	The handle to be freed.

Usage:

`SQLFreeHandle()` is used to free handles for environments, connections, statements, and descriptors.

An application should not use a handle after it has been freed; DB2 CLI does not check the validity of a handle in a function call.

Return codes:
- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

If `SQLFreeHandle()` returns SQL_ERROR, the handle is still valid.

Diagnostics:

Table 68. SQLFreeHandle SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The *HandleType* argument was SQL_HANDLE_DBC, and the communication link between DB2 CLI and the data source to which it was trying to connect failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.

Table 68. SQLFreeHandle SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The *HandleType* argument was SQL_HANDLE_ENV, and at least one connection was in an allocated or connected state. `SQLDisconnect()` and `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DBC must be called for each connection before calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_ENV. The *HandleType* argument was SQL_HANDLE_DBC, and the function was called before calling `SQLDisconnect()` for the connection. The *HandleType* argument was SQL_HANDLE_STMT; an asynchronously executing function was called on the statement handle; and the function was still executing when this function was called. The *HandleType* argument was SQL_HANDLE_STMT; `SQLExecute()` or `SQLExecDirect()` was called with the statement handle, and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. (DM) All subsidiary handles and other resources were not released before `SQLFreeHandle()` was called.
HY013	Unexpected memory handling error.	The *HandleType* argument was SQL_HANDLE_STMT or SQL_HANDLE_DESC, and the function call could not be processed because the underlying memory objects could not be accessed, possibly because of low memory conditions.
HY017	Invalid use of an automatically allocated descriptor handle.	The *Handle* argument was set to the handle for an automatically allocated descriptor or an implementation descriptor.

Restrictions:

None.

Example:

```
/* free the statement handle */
cliRC = SQLFreeHandle(SQL_HANDLE_STMT, hstmt2);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_STMT,
                                       hstmt2,
                                       cliRC,
                                       henv,
                                       hdbc,
                                       pOutSqlrc,
                                       outMsg,
                                       "SQLFreeHandle");
/* ... */
/* free the database handle */
cliRC = SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_DBC,
                                       hdbc,
                                       cliRC,
                                       henv,
                                       hdbc,
                                       pOutSqlrc,
                                       outMsg,
```

```
                                        "SQLFreeHandle");

/* free the environment handle */
cliRC = SQLFreeHandle(SQL_HANDLE_ENV, henv);
SRV_HANDLE_CHECK_SETTING_SQLRC_AND_MSG(SQL_HANDLE_ENV,
                                        henv,
                                        cliRC,
                                        henv,
                                        hdbc,
                                        pOutSqlrc,
                                        outMsg,
                                        "SQLFreeHandle");
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLCancel function (CLI) - Cancel statement" on page 49

Related samples:

- "tbmod.c -- How to modify table data"
- "tbread.c -- How to read data from tables"

SQLFreeStmt function (CLI) - Free (or reset) a statement handle

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLFreeStmt()` ends processing on the statement referenced by the statement handle. Use this function to:

- Close a cursor and discard all pending results
- Disassociate (reset) parameters from application variables and LOB file references
- Unbind columns from application variables and LOB file references
- Drop the statement handle and free the DB2 CLI resources associated with the statement handle.

`SQLFreeStmt()` is called after executing an SQL statement and processing the results.

Syntax:

```
SQLRETURN   SQLFreeStmt (SQLHSTMT          StatementHandle,   /* hstmt */
                         SQLUSMALLINT      Option);           /* fOption */
```

Function arguments:

Table 69. SQLFreeStmt arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle

Table 69. SQLFreeStmt arguments (continued)

Data type	Argument	Use	Description
SQLUSMALLINT	*Option*	input	Option which specifies the manner of freeing the statement handle. The option must have one of the following values: • SQL_CLOSE • SQL_DROP • SQL_UNBIND • SQL_RESET_PARAMS

Usage:

`SQLFreeStmt()` can be called with the following options:

SQL_CLOSE The cursor (if any) associated with the statement handle (*StatementHandle*) is closed and all pending results are discarded. The application can reopen the cursor by calling `SQLExecute()` with the same or different values in the application variables (if any) that are bound to *StatementHandle*. The cursor name is retained until the statement handle is dropped or a subsequent call to `SQLGetCursorName()` is successful. If no cursor has been associated with the statement handle, this option has no effect (no warning or error is generated).

`SQLCloseCursor()` can also be used to close a cursor.

SQL_DROP DB2 CLI resources associated with the input statement handle are freed, and the handle is invalidated. The open cursor, if any, is closed and all pending results are discarded.

This option has been replaced with a call to `SQLFreeHandle()` with the *HandleType* set to SQL_HANDLE_STMT. Although this version of DB2 CLI continues to support this option, we recommend that you begin using `SQLFreeHandle()` in your DB2 CLI programs so that they conform to the latest standards.

SQL_UNBIND

Sets the SQL_DESC_COUNT field of the ARD (Application Row Descriptor) to 0, releasing all column buffers bound by `SQLBindCol()` or `SQLBindFileToCol()` for the given *StatementHandle*. This does not unbind the bookmark column; to do that, the SQL_DESC_DATA_PTR field of the ARD for the bookmark column is set to NULL. Note that if this operation is performed on an explicitly allocated descriptor that is shared by more than one statement, the operation will affect the bindings of all statements that share the descriptor.

SQL_RESET_PARAMS

Sets the SQL_DESC_COUNT field of the APD (Application Parameter Descriptor) to 0, releasing all parameter buffers set by `SQLBindParameter()` or `SQLBindFileToParam()` for the given *StatementHandle*. Note that if this operation is performed on an explicitly allocated descriptor that is shared by more than one statement, this operation will affect the bindings of all the statements that share the descriptor.

`SQLFreeStmt()` has no effect on LOB locators, call `SQLExecDirect()` with the FREE LOCATOR statement to free a locator.

It is possible to reuse a statement handle to execute a different statement:

- If the handle was associated with a query, catalog function or SQLGetTypeInfo(), you must close the cursor.
- If the handle was bound with a different number or type of parameters, the parameters must be reset.
- If the handle was bound with a different number or type of column bindings, the columns must be unbound.

Alternatively you may drop the statement handle and allocate a new one.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

SQL_SUCCESS_WITH_INFO is not returned if *Option* is set to SQL_DROP, as there would be no statement handle to use when SQLGetDiagRec() or SQLGetDiagField() is called.

Diagnostics:

Table 70. SQLFreeStmt SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation.
HY092	Option type out of range.	The value specified for the argument *Option* was not SQL_CLOSE, SQL_DROP, SQL_UNBIND, or SQL_RESET_PARAMS.
HY506	Error closing a file.	Error encountered while trying to close a temporary file.

Authorization:

None.

Example:

```
/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_UNBIND);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
  return 1;
}

/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_RESET_PARAMS);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
```

```
{
  return 1;
}

/* free the statement handle */
cliRC = SQLFreeStmt(hstmt, SQL_CLOSE);
rc = HandleInfoPrint(SQL_HANDLE_STMT, hstmt, cliRC, __LINE__, __FILE__);
if (rc != 0)
{
  return 1;
}
```

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "LOB usage in ODBC applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLCloseCursor function (CLI) - Close cursor and discard pending results" on page 52
- "SQLFreeHandle function (CLI) - Free handle resources" on page 140
- "SQLGetTypeInfo function (CLI) - Get data type information" on page 224
- "SQLSetCursorName function (CLI) - Set cursor name" on page 272

Related samples:

- "utilcli.c -- Utility functions used by DB2 CLI samples"

SQLGetConnectAttr function (CLI) - Get current attribute setting

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLGetConnectAttr()` returns the current setting of a connection attribute.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetConnectAttrW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetConnectAttr(SQLHDBC          ConnectionHandle,
                              SQLINTEGER       Attribute,
                              SQLPOINTER       ValuePtr,
                              SQLINTEGER       BufferLength,
                              SQLINTEGER       *StringLengthPtr);
```

Function arguments:

Table 71. SQLGetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLINTEGER	*Attribute*	input	*Attribute* to retrieve.
SQLPOINTER	*ValuePtr*	output	A pointer to memory in which to return the current value of the attribute specified by *Attribute*.
SQLINTEGER	*BufferLength*	input	• If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*. • If *ValuePtr* is a pointer, but not to a string, then *BufferLength* should have the value SQL_IS_POINTER. • If the value in **ValuePtr* is a Unicode string the *BufferLength* argument must be an even number.
SQLINTEGER *	*StringLengthPtr*	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **ValuePtr*. If *ValuePtr* is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than BufferLength minus the length of the null-termination character, the data in **ValuePtr* is truncated to *BufferLength* minus the length of the null-termination character and is null-terminated by DB2 CLI.

Usage:

If *Attribute* specifies an attribute that returns a string, *ValuePtr* must be a pointer to a buffer for the string. The maximum length of the string, including the null termination character, will be *BufferLength* bytes.

Depending on the attribute, an application does not need to establish a connection prior to calling `SQLGetConnectAttr()`. However, if `SQLGetConnectAttr()` is called and the specified attribute does not have a default and has not been set by a prior call to `SQLSetConnectAttr()`, `SQLGetConnectAttr()` will return SQL_NO_DATA.

If *Attribute* is SQL_ATTR_TRACE or SQL_ATTR_TRACEFILE, *ConnectionHandle* does not have to be valid, and `SQLGetConnectAttr()` will not return SQL_ERROR if *ConnectionHandle* is invalid. These attributes apply to all connections. `SQLGetConnectAttr()` will return SQL_ERROR if another argument is invalid.

While an application can set statement attributes using `SQLSetConnectAttr()`, an application cannot use `SQLGetConnectAttr()` to retrieve statement attribute values; it must call `SQLGetStmtAttr()` to retrieve the setting of statement attributes.

The SQL_ATTR_AUTO_IPD connection attribute can be returned by a call to `SQLGetConnectAttr()`, but cannot be set by a call to `SQLSetConnectAttr()`.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 72. SQLGetConnectAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The data returned in **ValuePtr* was truncated to be *BufferLength* minus the length of a null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	An *Attribute* value was specified that required an open connection.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	`SQLBrowseConnect()` was called for the *ConnectionHandle* and returned SQL_NEED_DATA. This function was called before `SQLBrowseConnect()` returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* was less than 0.
HY092	Option type out of range.	The value specified for the argument *Attribute* was not valid.
HYC00	Driver not capable.	The value specified for the argument *Attribute* was a valid connection or statement attribute for the version of the DB2 CLI driver, but was not supported by the data source.

Restrictions:

None.

Example:

```
SQLINTEGER autocommit;

/* ... */

/* get the current setting for the AUTOCOMMIT attribute */
cliRC = SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, &autocommit, 0, NULL);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Connection attributes (CLI) list" on page 326
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLGetConnectOption function (CLI) - Return current setting of a connect option

Deprecated:

Note:

In ODBC version 3, `SQLGetConnectOption()` has been deprecated and replaced with `SQLGetConnectAttr()`.

Although this version of DB2 CLI continues to support `SQLGetConnectOption()`, we recommend that you begin using `SQLGetConnectAttr()` in your DB2 CLI programs so that they conform to the latest standards.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetConnectOptionW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Migrating to the new function

The statement:

```
SQLGetConnectOption(hdbc, SQL_ATTR_AUTOCOMMIT, pvAutoCommit);
```

for example, would be rewritten using the new function as:

```
SQLGetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, pvAutoCommit,
                  SQL_IS_POINTER, NULL);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLGetConnectAttr function (CLI) - Get current attribute setting" on page 146
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLGetCursorName function (CLI) - Get cursor name

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLGetCursorName()` returns the cursor name associated with the input statement handle. If a cursor name was explicitly set by calling `SQLSetCursorName()`, this name will be returned; otherwise, an implicitly generated name will be returned.

Unicode Equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetCursorNameW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetCursorName (
               SQLHSTMT         StatementHandle,  /* hstmt */
               SQLCHAR          *CursorName,      /* szCursor */
               SQLSMALLINT      BufferLength,     /* cbCursorMax */
               SQLSMALLINT      *NameLengthPtr);  /* pcbCursor */
```

Function arguments:

Table 73. SQLGetCursorName arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLCHAR *	*CursorName*	output	Cursor name
SQLSMALLINT	*BufferLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CursorName*.
SQLSMALLINT *	*NameLengthPtr*	output	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return for *CursorName*.

Usage:

`SQLGetCursorName()` will return the cursor name set explicitly with `SQLSetCursorName()`, or if no name was set, it will return the cursor name internally generated by DB2 CLI. If `SQLGetCursorName()` is called before a statement has been prepared on the input statement handle, an error will result. The internal cursor name is generated on a statement handle the first time dynamic SQL is prepared on the statement handle, not when the handle is allocated.

If a name is set explicitly using `SQLSetCursorName()`, this name will be returned until the statement is dropped, or until another explicit name is set.

Internally generated cursor names always begin with SQLCUR or SQL_CUR. Cursor names are always 18 SQLCHAR or SQLWCHAR elements or less, and are always unique within a connection.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 74. SQLGetCursorName SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The cursor name returned in *CursorName* was longer than the value in *BufferLength*, and is truncated to *BufferLength* - 1 bytes. The argument *NameLengthPtr* contains the length of the full cursor name available for return. The function returns SQL_SUCCESS_WITH_INFO.

Table 74. SQLGetCursorName SQLSTATEs (continued)

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called For the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* is less than 0.

Restrictions:

ODBC generated cursor names start with SQL_CUR, DB2 CLI generated cursor names start with SQLCUR, and X/Open CLI generated cursor names begin with either SQLCUR or SQL_CUR.

Example:

```
SQLCHAR cursorName[20];

/* ... */

/* get the cursor name of the SELECT statement */
cliRC = SQLGetCursorName(hstmtSelect, cursorName, 20, &cursorLen);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLSetCursorName function (CLI) - Set cursor name" on page 272
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spserver.c -- Definition of various types of stored procedures"
- "tbmod.c -- How to modify table data"

SQLGetData function (CLI) - Get data from a column

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLGetData() retrieves data for a single column in the current row of the result set. This is an alternative to SQLBindCol(), which is used to transfer data directly into application variables or LOB locators on each SQLFetch() or SQLFetchScroll() call. An application can either bind LOBs with SQLBindCol() or use SQLGetData() to retrieve LOBs, but both methods cannot be used together. SQLGetData() can also be used to retrieve large data values in pieces.

SQLFetch() or SQLFetchScroll() must be called before SQLGetData().

After calling SQLGetData() for each column, SQLFetch() or SQLFetchScroll() is called to retrieve the next row.

Syntax:

```
SQLRETURN   SQLGetData       (
               SQLHSTMT          StatementHandle,     /* hstmt */
               SQLUSMALLINT      ColumnNumber,        /* icol */
               SQLSMALLINT       TargetType,          /* fCType */
               SQLPOINTER        TargetValuePtr,      /* rgbValue */
               SQLLEN            BufferLength,        /* cbValueMax */
               SQLLEN            *StrLen_or_IndPtr);  /* pcbValue */
```

Function arguments:

Table 75. SQLGetData arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLUSMALLINT	*ColumnNumber*	input	Column number for which the data retrieval is requested. Result set columns are numbered sequentially from left to right. • Column numbers start at 1 if bookmarks are not used (SQL_ATTR_USE_BOOKMARKS statement attribute set to SQL_UB_OFF). • Column numbers start at 0 if bookmarks are used (the statement attribute set to SQL_UB_ON or SQL_UB_VARIABLE).

Table 75. SQLGetData arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*TargetType*	input	The C data type of the column identifier by *ColumnNumber*. The following types are supported: • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DECIMAL_IBM • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_NUMERIC [a] • SQL_C_SBIGINT • SQL_C_SHORT • SQL_C_TYPE_DATE • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TINYINT • SQL_C_UBIGINT • SQL_C_UTINYINT • SQL_C_WCHAR Specifying SQL_ARD_TYPE results in the data being converted to the data type specified in the SQL_DESC_CONCISE_TYPE field of the ARD. Specifying SQL_C_DEFAULT results in the data being converted to its default C data type.
SQLPOINTER	*TargetValuePtr*	output	Pointer to buffer where the retrieved column data is to be stored.
SQLLEN	*BufferLength*	input	Maximum size of the buffer pointed to by *TargetValuePtr*. This value is ignored when the driver returns fixed-length data.
SQLLEN *	*StrLen_or_IndPtr*	output	Pointer to value which indicates the number of bytes DB2 CLI has available to return in the *TargetValuePtr* buffer. If the data is being retrieved in pieces, this contains the number of bytes still remaining. The value is SQL_NULL_DATA if the data value of the column is null. If this pointer is NULL and `SQLFetch()` has obtained a column containing null data, then this function will fail because it has no means of reporting this. If `SQLFetch()` has fetched a column containing binary data, then the pointer to *StrLen_or_IndPtr* must not be NULL or this function will fail because it has no other means of informing the application about the length of the data retrieved in the *TargetValuePtr* buffer.

Note: DB2 CLI will provide some performance enhancement if *TargetValuePtr* is placed consecutively in memory after *StrLen_or_IndPtr*

Usage:

Different DB2 data sources have different restrictions on how SQLGetData() can be used. For an application to be sure about the functional capabilities of this function, it should call SQLGetInfo() with any of the following SQL_GETDATA_EXTENSIONS options:

- SQL_GD_ANY_COLUMN: If this option is returned, SQLGetData() can be called for any unbound column, including those before the last bound column. All DB2 data sources support this feature.
- SQL_GD_ANY_ORDER: If this option is returned, SQLGetData() can be called for unbound columns in any order. All DB2 data sources support this feature.
- SQL_GD_BLOCK: If this option if returned by SQLGetInfo() for the SQL_GETDATA_EXTENSIONS *InfoType* argument, then the driver will support calls to SQLGetData() when the rowset size is greater than 1. The application can also call SQLSetPos() with the SQL_POSITION option to position the cursor on the correct row before calling SQLGetData(). At least DB2 for Unix and Windows data sources support this feature.
- SQL_GD_BOUND: If this option is returned, SQLGetData() can be called for bound columns as well as unbound columns. DB2 Database for Linux, UNIX, and Windows does not currently support this feature.

SQLGetData() can also be used to retrieve long columns if the C data type (*TargetType*) is SQL_C_CHAR, SQL_C_BINARY, SQL_C_DBCHAR, SQL_C_WCHAR, or if *TargetType* is SQL_C_DEFAULT and the column type denotes a binary or character string.

Upon each SQLGetData() call, if the data available for return is greater than or equal to *BufferLength*, truncation occurs. Truncation is indicated by a function return code of SQL_SUCCESS_WITH_INFO coupled with a SQLSTATE denoting data truncation. The application can call SQLGetData() again, with the same ColumnNumber value, to get subsequent data from the same unbound column starting at the point of truncation. To obtain the entire column, the application repeats such calls until the function returns SQL_SUCCESS. The next call to SQLGetData() returns SQL_NO_DATA_FOUND.

Although SQLGetData() can be used for the sequential retrieval of LOB column data, use the DB2 CLI LOB functions if only a portion of the LOB data or a few sections of the LOB column data are needed:

1. Bind the column to a LOB locator.
2. Fetch the row.
3. Use the locator in a SQLGetSubString() call, to retrieve the data in pieces (SQLGetLength() and SQLGetPosition() might also be required in order to determine the values of some of the arguments).
4. Repeat step 2.

Truncation is also affected by the SQL_ATTR_MAX_LENGTH statement attribute. The application can specify that truncation is not to be reported by calling SQLSetStmtAttr() with SQL_ATTR_MAX_LENGTH and a value for the maximum length to return for any one column, and by allocating a *TargetValuePtr* buffer of the same size (plus the null-terminator). If the column data is larger than the set maximum length, SQL_SUCCESS will be returned and the maximum length, not the actual length will be returned in *StrLen_or_IndPtr*.

To discard the column data part way through the retrieval, the application can call SQLGetData() with *ColumnNumber* set to the next column position of interest. To discard data that has not been retrieved for the entire row, the application should call SQLFetch() to advance the cursor to the next row; or, if it does not want any

more data from the result set, the application can close the cursor by calling SQLCloseCursor() or SQLFreeStmt() with the SQL_CLOSE or SQL_DROP option.

The *TargetType* input argument determines the type of data conversion (if any) needed before the column data is placed into the storage area pointed to by *TargetValuePtr*.

For SQL graphic column data:

- The length of the *TargetValuePtr* buffer (*BufferLength*) should be a multiple of 2. The application can determine the SQL data type of the column by first calling SQLDescribeCol() or SQLColAttribute().
- The pointer to *StrLen_or_IndPtr* must not be NULL since DB2 CLI will be storing the number of octets stored in *TargetValuePtr*.
- If the data is to be retrieved in piecewise fashion, DB2 CLI will attempt to fill *TargetValuePtr* to the nearest multiple of two octets that is still less than or equal to *BufferLength*. This means if *BufferLength* is not a multiple of two, the last byte in that buffer will be untouched; DB2 CLI will not split a double-byte character.

The content returned in *TargetValuePtr* is always null-terminated unless the column data to be retrieved is binary, or if the SQL data type of the column is graphic (DBCS) and the C buffer type is SQL_C_CHAR. If the application is retrieving the data in multiple chunks, it should make the proper adjustments (for example, strip off the null-terminator before concatenating the pieces back together assuming the null termination environment attribute is in effect).

Truncation of numeric data types is reported as a warning if the truncation involves digits to the right of the decimal point. If truncation occurs to the left of the decimal point, an error is returned (refer to the diagnostics section).

With the exception of scrollable cursors, applications that use SQLFetchScroll() to retrieve data should call SQLGetData() only when the rowset size is 1 (equivalent to issuing SQLFetch()). SQLGetData() can only retrieve column data for a row where the cursor is currently positioned.

Using SQLGetData() with Scrollable Cursors

SQLGetData() can also be used with scrollable cursors. You can save a pointer to any row in the result set with a bookmark. The application can then use that bookmark as a relative position to retrieve a rowset of information.

Once you have positioned the cursor to a row in a rowset using SQLSetPos(), you can obtain the bookmark value from column 0 using SQLGetData(). In most cases you will not want to bind column 0 and retrieve the bookmark value for every row, but use SQLGetData() to retrieve the bookmark value for the specific row you require.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND
- SQL_NO_TOTAL

SQL_NO_DATA_FOUND is returned when the preceding SQLGetData() call has retrieved all of the data for this column.

SQL_SUCCESS is returned if a zero-length string is retrieved by SQLGetData(). If this is the case, StrLen_or_IndPtr will contain 0, and TargetValuePtr will contain a null terminator.

SQL_NO_TOTAL is returned as the length when truncation occurs if the DB2 CLI configuration keyword StreamGetData is set to 1 and DB2 CLI cannot determine the number of bytes still available to return in the output buffer.

If the preceding call to SQLFetch() failed, SQLGetData() should not be called since the result is undefined.

Diagnostics:

Table 76. SQLGetData SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	Data returned for the specified column (ColumnNumber) was truncated. String or numeric values are right truncated. SQL_SUCCESS_WITH_INFO is returned.
07006	Invalid conversion.	The data value cannot be converted to the C data type specified by the argument *TargetType*. The function has been called before for the same *ColumnNumber* value but with a different *TargetType* value.
07009	Invalid descriptor index.	The value specified for *ColumnNumber* was equal to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF. The value specified for the argument *ColumnNumber* was greater than the number of columns in the result set.
22002	Invalid output or indicator buffer specified.	The pointer value specified for the argument StrLen_or_IndPtr was a null pointer and the value of the column is null. There is no means to report SQL_NULL_DATA.
22003	Numeric value out of range.	Returning the numeric value (as numeric or string) for the column would have caused the whole part of the number to be truncated.
22005	Error in assignment.	A returned value was incompatible with the data type denoted by the argument *TargetType*.
22007	Invalid datetime format.	Conversion from character a string to a datetime format was indicated, but an invalid string representation or value was specified, or the value was an invalid date.
22008	Datetime field overflow.	Datetime field overflow occurred; for example, an arithmetic operation on a date or timestamp has a result that is not within the valid range of dates, or a datetime value cannot be assigned to a bound variable because it is too small.
24000	Invalid cursor state.	The previous SQLFetch() resulted in SQL_ERROR or SQL_NO_DATA found; as a result, the cursor is not positioned on a row.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.

Table 76. SQLGetData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	*TargetType* was not a valid data type or SQL_C_DEFAULT.
HY010	Function sequence error.	The specified *StatementHandle* was not in a cursor positioned state. The function was called without first calling `SQLFetch()`. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called For the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY011	Operation invalid at this time.	Calls to `SQLGetData()` for previously accessed LOB columns are not allowed. Refer to AllowGetDataLOBReaccess CLI/ODBC configuration keyword for more information.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value of the argument *BufferLength* is less than 0 and the argument *TargetType* is SQL_C_CHAR, SQL_C_BINARY, SQL_C_DBCHAR or (SQL_C_DEFAULT and the default type is one of SQL_C_CHAR, SQL_C_BINARY, or SQL_C_DBCHAR).
HYC00	Driver not capable.	The SQL data type for the specified data type is recognized but not supported by DB2 CLI. The requested conversion from the SQL data type to the application data *TargetType* cannot be performed by DB2 CLI or the data source. The column was bound using `SQLBindFileToCol()`.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* use SQLGetData to get the results */
/* get data from column 1 */
cliRC = SQLGetData(hstmt,
                   1,
                   SQL_C_SHORT,
                   &deptnumb.val,
                   0,
                   &deptnumb.ind);
```

```
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* get data from column 2 */
cliRC = SQLGetData(hstmt,
                   2,
                   SQL_C_CHAR,
                   location.val,
                   15,
                   &location.ind);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dtlob.c -- How to read and write LOB data"
- "tbread.c -- How to read data from tables"

SQLGetDataLinkAttr function (CLI) - Get DataLink attribute value

Purpose:

Specification:	DB2 CLI 5.2		ISO CLI

Return the current value of an attribute of a datalink value.

Syntax:

```
SQLRETURN   SQLGetDataLinkAttr (
               SQLHSTMT         StatementHandle,    /* hStmt */
               SQLSMALLINT      Attribute,          /* fAttrType */
               SQLCHAR          *DataLink,          /* *pDataLink */
               SQLINTEGER       DataLinkLength,     /* cbDataLink */
               SQLPOINTER       *ValuePtr,          /* pAttribute */
               SQLINTEGER       BufferLength,       /* cbAttributeMax */
               SQLINTEGER       *StringLengthPtr);  /* *pcbAttribute */
```

Function arguments:

Table 77. SQLGetDataLinkAttr Arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Used only for diagnostic reporting.

Table 77. SQLGetDataLinkAttr Arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*Attribute*	input	Identifies the attribute of the DataLink that is to be extracted. Possible values are: • SQL_ATTR_DATALINK_COMMENT • SQL_ATTR_DATALINK_LINKTYPE • SQL_ATTR_DATALINK_URLCOMPLETE (complete URL to access a file) • SQL_ATTR_DATALINK_URLPATH (to access a file within a file server) • SQL_ATTR_DATALINK_URLPATHONLY (file path only) • SQL_ATTR_DATALINK_URLSCHEME • SQL_ATTR_DATALINK_URLSERVER
SQLCHAR *	*DataLink*	input	The DATALINK value from which the attribute is to be extracted.
SQLINTEGER	*DataLinkLength*	input	The length of the *DataLink* value. If the *DataLink* argument contains a null-terminated string, a value of SQL_NTS can be passed for *DataLinkLength*.
SQLPOINTER *	*ValuePtr*	output	A pointer to memory in which to return the value of the attribute specified by *Attribute*.
SQLINTEGER	*BufferLength*	input	The amount of storage available at *ValuePtr* to hold the return value.
SQLINTEGER *	*StringLength*	output	A pointer to a buffer in which to return the total number of bytes (excluding the null-termination character) available to return in **Attribute*. If *Attribute* is a null pointer, no length is returned. If the number of bytes available to return is greater than BufferLength minus the length of the null-termination character, then SQLSTATE HY090 is returned.

Usage:

The function is used with a DATALINK value that was retrieved from the database or built using `SQLBuildDataLink()`. The *AttrType* value determines the attribute from the DATALINK value that is returned. The maximum length of the string, including the null termination character, will be *BufferLength* bytes.

Return codes:

- SQL_SUCCESS
- SQL_NO_DATA
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 78. SQLGetDataLinkAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.

Table 78. SQLGetDataLinkAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
01004	Data truncated.	The data returned in **ValuePtr* was truncated to be *BufferLength* minus the length of the null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	The value specified for the argument **DataLink* was a null pointer or was not valid.
HY090	Invalid string or buffer length.	The value specified for the argument *BufferLength* was less than 0 or the values specified for the argument *DataLinkLength* was less than 0 and not equal to SQL_NTS.
HY092	Option type out of range.	The value specified for the argument *AttrType* was not valid.

Restrictions:

DB2 Data Links Manager is no longer supported for DB2 on Linux, UNIX and Windows. Check your server for support.

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBuildDataLink function (CLI) - Build DATALINK value" on page 41

SQLGetDescField function (CLI) - Get single field settings of descriptor record

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLGetDescField()` returns the current settings of a single field of a descriptor record.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetDescFieldW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetDescField (
               SQLHDESC          DescriptorHandle,
               SQLSMALLINT       RecNumber,
               SQLSMALLINT       FieldIdentifier,
```

```
SQLPOINTER        ValuePtr,           /* Value */
SQLINTEGER        BufferLength,
SQLINTEGER        *StringLengthPtr);  /* *StringLength */
```

Function arguments:

Table 79. SQLGetDescField arguments

Data type	Argument	Use	Description
SQLHDESC	*DescriptorHandle*	input	Descriptor handle.
SQLSMALLINT	*RecNumber*	input	Indicates the descriptor record from which the application seeks information. Descriptor records are numbered from 0, with record number 0 being the bookmark record. If the *FieldIdentifier* argument indicates a field of the descriptor header record, *RecNumber* must be 0. If *RecNumber* is less than SQL_DESC_COUNT, but the row does not contain data for a column or parameter, a call to `SQLGetDescField()` will return the default values of the fields.
SQLSMALLINT	*FieldIdentifier*	input	Indicates the field of the descriptor whose value is to be returned.
SQLPOINTER	*ValuePtr*	output	Pointer to a buffer in which to return the descriptor information. The data type depends on the value of *FieldIdentifier*.
SQLINTEGER	*BufferLength*	input	• If *ValuePtr* points to a character string, this argument should be the length of **ValuePtr*. • If *ValuePtr* is a pointer, but not to a string, then *BufferLength* should have the value SQL_IS_POINTER. • If the value in **ValuePtr* is of a Unicode data type the *BufferLength* argument must be an even number.
SQLSMALLINT *	*StringLengthPtr*	output	Pointer to the total number of bytes (excluding the number of bytes required for the null termination character) available to return in **ValuePtr*.

Usage:

An application can call `SQLGetDescField()` to return the value of a single field of a descriptor record. A call to `SQLGetDescField()` can return the setting of any field in any descriptor type, including header fields, record fields, and bookmark fields. An application can obtain the settings of multiple fields in the same or different descriptors, in arbitrary order, by making repeated calls to `SQLGetDescField()`. `SQLGetDescField()` can also be called to return DB2 CLI defined descriptor fields.

For performance reasons, an application should not call `SQLGetDescField()` for an IRD before executing a statement. Calling `SQLGetDescField()` in this case causes the CLI driver to describe the statement, resulting in an extra network flow. When deferred prepare is on and `SQLGetDescField()` is called, you lose the benefit of deferred prepare because the statement must be prepared at the server to obtain describe information.

The settings of multiple fields that describe the name, data type, and storage of column or parameter data can also be retrieved in a single call to `SQLGetDescRec()`. `SQLGetStmtAttr()` can be called to return the value of a single field in the descriptor header that has an associated statement attribute.

When an application calls SQLGetDescField() to retrieve the value of a field that is undefined for a particular descriptor type, the function returns SQLSTATE HY091 (Invalid descriptor field identifier). When an application calls SQLGetDescField() to retrieve the value of a field that is defined for a particular descriptor type, but has no default value and has not been set yet, the function returns SQL_SUCCESS but the value returned for the field is undefined. Refer to the list of initialization values of descriptor fields for any default values which may exist.

The SQL_DESC_ALLOC_TYPE header field is available as read-only. This field is defined for all types of descriptors.

Each of these fields is defined either for the IRD only, or for both the IRD and the IPD.

```
SQL_DESC_AUTO_UNIQUE_VALUE    SQL_DESC_LITERAL_SUFFIX
SQL_DESC_BASE_COLUMN_NAME     SQL_DESC_LOCAL_TYPE_NAME
SQL_DESC_CASE_SENSITIVE       SQL_DESC_SCHEMA_NAME
SQL_DESC_CATALOG_NAME         SQL_DESC_SEARCHABLE
SQL_DESC_DISPLAY_SIZE         SQL_DESC_TABLE_NAME
SQL_DESC_FIXED_PREC_SCALE     SQL_DESC_TYPE_NAME
SQL_DESC_LABEL                SQL_DESC_UNSIGNED
SQL_DESC_LITERAL_PREFIX       SQL_DESC_UPDATABLE
```

Refer to the list of descriptor *FieldIdentifier* values for more information about the above fields.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_NO_DATA
- SQL_INVALID_HANDLE

SQL_NO_DATA is returned if *RecNumber* is greater than the number of descriptor records.

SQL_NO_DATA is returned if *DescriptorHandle* is an IRD handle and the statement is in the prepared or executed state, but there was no open cursor associated with it.

Diagnostics:

Table 80. SQLGetDescField SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The buffer **ValuePtr* was not large enough to return the entire descriptor field, so the field was truncated. The length of the untruncated descriptor field is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)

Table 80. SQLGetDescField SQLSTATEs (continued)

SQLSTATE	Description	Explanation
07009	Invalid descriptor index.	The value specified for the *RecNumber* argument was less than 1, the SQL_ATTR_USE_BOOKMARKS statement attribute was SQL_UB_OFF, and the field was not a header field or a DB2 CLI defined field. The *FieldIdentifier* argument was a record field, and the *RecNumber* argument was 0. The *RecNumber* argument was less than 0, and the field was not a header field or a DB2 CLI defined field.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY007	Associated statement is not prepared.	*DescriptorHandle* was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error.	*DescriptorHandle* was associated with a StatementHandle for which an asynchronously executing function (not this one) was called and was still executing when this function was called. *DescriptorHandle* was associated with a *StatementHandle* for which `SQLExecute()` or `SQLExecDirect()` was called and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY021	Inconsistent descriptor information.	The descriptor information checked during a consistency check was not consistent.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
HY091	Invalid descriptor field identifier.	*FieldIdentifier* was undefined for the *DescriptorHandle*. The value specified for the *RecNumber* argument was greater than the value in the SQL_DESC_COUNT field.

Restrictions:

None.

Example:

```
/* see how the field SQL_DESC_PARAMETER_TYPE is set */
cliRC = SQLGetDescField(hIPD,
                        1, /* look at the parameter */
                        SQL_DESC_PARAMETER_TYPE,
                        &descFieldParameterType, /* result */
                        SQL_IS_SMALLINT,
```

```
                        NULL);

/* ... */

  /* see how the descriptor record field SQL_DESC_TYPE_NAME is set */
  rc = SQLGetDescField(hIRD,
                       (SQLSMALLINT)colCount,
                       SQL_DESC_TYPE_NAME, /* record field */
                       descFieldTypeName, /* result */
                       25,
                       NULL);
  /* ... */

  /* see how the descriptor record field SQL_DESC_LABEL is set */
  rc = SQLGetDescField(hIRD,
                       (SQLSMALLINT)colCount,
                       SQL_DESC_LABEL, /* record field */
                       descFieldLabel, /* result */
                       25,
                       NULL);
```

Related concepts:

- "Consistency checks for descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Descriptor FieldIdentifier argument values (CLI)" on page 367
- "Descriptor header and record field initialization values (CLI)" on page 378
- "SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record" on page 164
- "SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute" on page 216
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbuse.c -- How to use a database"

SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

`SQLGetDescRec()` returns the current settings of multiple fields of a descriptor record. The fields returned describe the name, data type, and storage of column or parameter data.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetDescRecW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetDescRec (
               SQLHDESC          DescriptorHandle,   /* hDesc */
               SQLSMALLINT       RecNumber,
               SQLCHAR           *Name,
               SQLSMALLINT       BufferLength,
               SQLSMALLINT       *StringLengthPtr,   /* *StringLength */
               SQLSMALLINT       *TypePtr,           /* *Type */
               SQLSMALLINT       *SubTypePtr,        /* *SubType */
               SQLLEN            *LengthPtr,         /* *Length */
               SQLSMALLINT       *PrecisionPtr,      /* *Precision */
               SQLSMALLINT       *ScalePtr,          /* *Scale */
               SQLSMALLINT       *NullablePtr);      /* *Nullable */
```

Function arguments:

Table 81. SQLGetDescRec arguments

Data type	Argument	Use	Description
SQLHDESC	*DescriptorHandle*	input	Descriptor handle.
SQLSMALLINT	*RecNumber*	input	Indicates the descriptor record from which the application seeks information. Descriptor records are numbered from 0, with record number 0 being the bookmark record. The *RecNumber* argument must be less than or equal to the value of SQL_DESC_COUNT. If *RecNumber* is less than SQL_DESC_COUNT, but the row does not contain data for a column or parameter, a call to `SQLGetDescRec()` will return the default values of the fields.
SQLCHAR *	*Name*	output	A pointer to a buffer in which to return the SQL_DESC_NAME field for the descriptor record.
SQLINTEGER	*BufferLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the **Name* buffer.
SQLSMALLINT *	*StringLengthPtr*	output	A pointer to a buffer in which to return the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) available to return in the *Name* buffer, excluding the null-termination character. If the number of SQLCHAR or SQLWCHAR elements was greater than or equal to *BufferLength*, the data in **Name* is truncated to *BufferLength* minus the length of a null-termination character, and is null terminated by DB2 CLI.
SQLSMALLINT *	*TypePtr*	output	A pointer to a buffer in which to return the value of the SQL_DESC_TYPE field for the descriptor record.
SQLSMALLINT *	*SubTypePtr*	output	For records whose type is SQL_DATETIME, this is a pointer to a buffer in which to return the value of the SQL_DESC_DATETIME_INTERVAL_CODE field.
SQLLEN *	*LengthPtr*	output	A pointer to a buffer in which to return the value of the SQL_DESC_OCTET_LENGTH field for the descriptor record.

Table 81. SQLGetDescRec arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	*PrecisionPtr*	output	A pointer to a buffer in which to return the value of the SQL_DESC_PRECISION field for the descriptor record.
SQLSMALLINT *	*ScalePtr*	output	A pointer to a buffer in which to return the value of the SQL_DESC_SCALE field for the descriptor record.
SQLSMALLINT *	*NullablePtr*	output	A pointer to a buffer in which to return the value of the SQL_DESC_NULLABLE field for the descriptor record.

Usage:

An application can call `SQLGetDescRec()` to retrieve the values of the following fields for a single column or parameter:
- SQL_DESC_NAME
- SQL_DESC_TYPE
- SQL_DESC_DATETIME_INTERVAL_CODE (for records whose type is SQL_DATETIME)
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_NULLABLE

`SQLGetDescRec()` does not retrieve the values for header fields.

An application can inhibit the return of a field's setting by setting the argument corresponding to the field to a null pointer. When an application calls `SQLGetDescRec()` to retrieve the value of a field that is undefined for a particular descriptor type, the function returns SQL_SUCCESS but the value returned for the field is undefined. For example, calling `SQLGetDescRec()` for the SQL_DESC_NAME or SQL_DESC_NULLABLE field of an APD or ARD will return SQL_SUCCESS but an undefined value for the field.

When an application calls `SQLGetDescRec()` to retrieve the value of a field that is defined for a particular descriptor type, but has no default value and has not been set yet, the function returns SQL_SUCCESS but the value returned for the field is undefined.

The values of fields can also be retrieved individually by a call to `SQLGetDescField()`.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_NO_DATA
- SQL_INVALID_HANDLE

SQL_NO_DATA is returned if *RecNumber* is greater than the number of descriptor records.

SQL_NO_DATA is returned if *DescriptorHandle* is an IRD handle and the statement in the prepared or executed state, but there was no open cursor associated with it.

Diagnostics:

Table 82. SQLGetDescRec SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The buffer **Name* was not large enough to return the entire descriptor field, so the field was truncated. The length of the untruncated descriptor field is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index.	The *RecNumber* argument was set to 0 and the *DescriptorHandle* argument was an IPD handle. The *RecNumber* argument was set to 0, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF. The *RecNumber* argument was less than 0.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY007	Associated statement is not prepared.	*DescriptorHandle* was associated with an IRD, and the associated statement handle was not in the prepared or executed state.
HY010	Function sequence error.	*DescriptorHandle* was associated with a *StatementHandle* for which an asynchronously executing function (not this one) was called and was still executing when this function was called. *DescriptorHandle* was associated with a *StatementHandle* for which `SQLExecute()` or `SQLExecDirect()` was called and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Restrictions:

None.

Example:

```
/* get multiple field settings of descriptor record */
rc = SQLGetDescRec(hIRD,
                   i,
                   colname,
                   sizeof(colname),
                   &namelen,
```

```
                    &type,
                    &subtype,
                    &width,
                    &precision,
                    &scale,
                    &nullable);

/* ... */

/* get the record/column value after setting */
rc = SQLGetDescRec(hARD,
                    i,
                    colname,
                    sizeof(colname),
                    &namelen,
                    &type,
                    &subtype,
                    &width,
                    &precision,
                    &scale,
                    &nullable);
```

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Descriptor header and record field initialization values (CLI)" on page 378
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" on page 160
- "SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data" on page 281
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbuse.c -- How to use a database"

SQLGetDiagField function (CLI) - Get a field of diagnostic data

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLGetDiagField()` returns the current value of a field of a diagnostic data structure, associated with a specific handle, that contains error, warning, and status information.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is

SQLGetDiagFieldW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetDiagField (
               SQLSMALLINT      HandleType,         /* fHandleType */
               SQLHANDLE        Handle,             /* hHandle */
               SQLSMALLINT      RecNumber,          /* iRecNumber */
               SQLSMALLINT      DiagIdentifier,     /* fDiagIdentifier */
               SQLPOINTER       DiagInfoPtr,        /* pDiagInfo */
               SQLSMALLINT      BufferLength,       /* cbDiagInfoMax */
               SQLSMALLINT      *StringLengthPtr);  /* *pcgDiagInfo */
```

Function arguments:

Table 83. SQLGetDiagField arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	A handle type identifier that describes the type of handle for which diagnostics are desired. Must be one of the following: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
SQLHANDLE	*Handle*	input	A handle for the diagnostic data structure, of the type indicated by *HandleType*.
SQLSMALLINT	*RecNumber*	input	Indicates the status record from which the application seeks information. Status records are numbered from 1. If the *DiagIdentifier* argument indicates any field of the diagnostics header record, *RecNumber* must be 0. If not, it should be greater than 0.
SQLSMALLINT	*DiagIdentifier*	input	Indicates the field of the diagnostic data structure whose value is to be returned. For more information, see "DiagIdentifier argument" on page 171.
SQLPOINTER	*DiagInfoPtr*	output	Pointer to a buffer in which to return the diagnostic information. The data type depends on the value of *DiagIdentifier*.

Table 83. SQLGetDiagField arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	*BufferLength*	input	If *DiagIdentifier* is ODBC-defined diagnostic: • If *DiagInfoPtr* points to a character string or binary buffer, *BufferLength* should be the length of **DiagInfoPtr*. • If **DiagInfoPtr* is an integer, *BufferLength* is ignored. • If **DiagInfoPtr* is a Unicode string, *BufferLength* must be an even number. If *DiagIdentifier* is a DB2 CLI diagnostic: • If **DiagInfoPtr* is a pointer to a character string, *BufferLength* is the number of bytes needed to store the string, or SQL_NTS. • If **DiagInfoPtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *BufferLength*. This places a negative value in *BufferLength*. • If **DiagInfoPtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER. • If **DiagInfoPtr* contains a fixed-length data type, then *BufferLength* is SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.
SQLSMALLINT *	*StringLengthPtr*	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the number of bytes required for the null-termination character, available to return in **DiagInfoPtr*, for character data. If the number of bytes available to return is greater than *BufferLength*, then the text in **DiagInfoPtr* is truncated to *BufferLength* minus the length of a null-termination character. This argument is ignored for non-character data.

Usage:

An application typically calls `SQLGetDiagField()` to accomplish one of three goals:

1. To obtain specific error or warning information when a function call has returned the SQL_ERROR or SQL_SUCCESS_WITH_INFO (or SQL_NEED_DATA for the `SQLBrowseConnect()` function) return codes.
2. To find out the number of rows in the data source that were affected when insert, delete, or update operations were performed with a call to `SQLExecute()`, `SQLExecDirect()`, `SQLBulkOperations()`, or `SQLSetPos()` (from the SQL_DIAG_ROW_COUNT header field), or to find out the number of rows that exist in the current open static scrollable cursor (from the SQL_DIAG_CURSOR_ROW_COUNT header field).
3. To determine which function was executed by a call to `SQLExecDirect()` or `SQLExecute()` (from the SQL_DIAG_DYNAMIC_FUNCTION and SQL_DIAG_DYNAMIC_FUNCTION_CODE header fields).

Any DB2 CLI function can post zero or more errors each time it is called, so an application can call SQLGetDiagField() after any function call. SQLGetDiagField() retrieves only the diagnostic information most recently associated with the diagnostic data structure specified in the *Handle* argument. If the application calls another function, any diagnostic information from a previous call with the same handle is lost.

An application can scan all diagnostic records by incrementing *RecNumber*, as long as SQLGetDiagField() returns SQL_SUCCESS. The number of status records is indicated in the SQL_DIAG_NUMBER header field. Calls to SQLGetDiagField() are non-destructive as far as the header and status records are concerned. The application can call SQLGetDiagField() again at a later time to retrieve a field from a record, as long as another function other than SQLGetDiagField(), SQLGetDiagRec(), or SQLError() has not been called in the interim, which would post records on the same handle.

An application can call SQLGetDiagField() to return any diagnostic field at any time, with the exception of SQL_DIAG_ROW_COUNT, which will return SQL_ERROR if *Handle* was not a statement handle on which an SQL statement had been executed. If any other diagnostic field is undefined, the call to SQLGetDiagField() will return SQL_SUCCESS (provided no other error is encountered), and an undefined value is returned for the field.

HandleType argument

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of *Handle*.

Some header and record fields cannot be returned for all types of handles: environment, connection, statement, and descriptor. Those handles for which a field is not applicable are indicated in the Header Field and Record Fields sections below.

No DB2 CLI-specific header diagnostic field should be associated with an environment handle.

DiagIdentifier argument

This argument indicates the identifier of the field desired from the diagnostic data structure. If *RecNumber* is greater than or equal to 1, the data in the field describes the diagnostic information returned by a function. If *RecNumber* is 0, the field is in the header of the diagnostic data structure, so it contains data pertaining to the function call that returned the diagnostic information, not the specific information. Refer to the list of header and record fields for the *DiagIdentifier* argument for further information.

Sequence of status records

Status records are placed in a sequence based upon row number and the type of the diagnostic.

If there are two or more status records, the sequence of the records is determined first by row number. The following rules apply to determining the sequence of errors by row:

- Records that do not correspond to any row appear in front of records that correspond to a particular row, since SQL_NO_ROW_NUMBER is defined to be -1.
- Records for which the row number is unknown appear in front of all other records, since SQL_ROW_NUMBER_UNKNOWN is defined to be -2.
- For all records that pertain to specific rows, records are sorted by the value in the SQL_DIAG_ROW_NUMBER field. All errors and warnings of the first row affected are listed, then all errors and warnings of the next row affected, and so on.

Within each row, or for all those records that do not correspond to a row or for which the row number is unknown, the first record listed is determined using a set of sorting rules. After the first record, the order of the other records affecting a row is undefined. An application cannot assume that errors precede warnings after the first record. Applications should scan the entire diagnostic data structure to obtain complete information on an unsuccessful call to a function.

The following rules are followed to determine the first record within a row. The record with the highest rank is the first record.

- **Errors**. Status records that describe errors have the highest rank. The following rules are followed to sort errors:
  - Records that indicate a transaction failure or possible transaction failure outrank all other records.
  - If two or more records describe the same error condition, then SQLSTATEs defined by the X/Open CLI specification (classes 03 through HZ) outrank ODBC- and driver-defined SQLSTATEs.
- **Implementation-defined No Data values**. Status records that describe DB2 CLI No Data values (class 02) have the second highest rank.
- **Warnings**. Status records that describe warnings (class 01) have the lowest rank. If two or more records describe the same warning condition, then warning SQLSTATEs defined by the X/Open CLI specification outrank ODBC- and driver-defined SQLSTATEs.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA

Diagnostics:

`SQLGetDiagField()` does not post error values for itself. It uses the following return values to report the outcome of its own execution:

- SQL_SUCCESS: The function successfully returned diagnostic information.
- SQL_SUCCESS_WITH_INFO: **DiagInfoPtr* was too small to hold the requested diagnostic field so the data in the diagnostic field was truncated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to **StringLengthPtr*.
- SQL_INVALID_HANDLE: The handle indicated by *HandleType* and *Handle* was not a valid handle.
- SQL_ERROR: One of the following occurred:
  - The *DiagIdentifier* argument was not one of the valid values.

  - The *DiagIdentifier* argument was SQL_DIAG_CURSOR_ROW_COUNT, SQL_DIAG_DYNAMIC_FUNCTION, SQL_DIAG_DYNAMIC_FUNCTION_CODE, or SQL_DIAG_ROW_COUNT, but *Handle* was not a statement handle.
  - The *RecNumber* argument was negative or 0 when *DiagIdentifier* indicated a field from a diagnostic record. *RecNumber* is ignored for header fields.
  - The value requested was a character string and *BufferLength* was less than zero.
- SQL_NO_DATA: *RecNumber* was greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL_NO_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

Restrictions:

None.

Related concepts:

- "Diagnostics in CLI applications overview" in *Call Level Interface Guide and Reference, Volume 1*
- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Header and record fields for the DiagIdentifier argument (CLI)" on page 385
- "SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record" on page 173
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLGetDiagRec()` returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information. Unlike `SQLGetDiagField()`, which returns one diagnostic field per call, `SQLGetDiagRec()` returns several commonly used fields of a diagnostic record: the SQLSTATE, native error code, and error message text.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetDiagRecW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetDiagRec (
               SQLSMALLINT       HandleType,       /* fHandleType */
               SQLHANDLE         Handle,           /* hHandle */
```

```
SQLSMALLINT       RecNumber,        /* iRecNumber */
SQLCHAR           *SQLState,        /* *pszSqlState */
SQLINTEGER        *NativeErrorPtr,  /* *pfNativeError */
SQLCHAR           *MessageText,     /* *pszErrorMsg */
SQLSMALLINT       BufferLength,     /* cbErrorMsgMax */
SQLSMALLINT       *TextLengthPtr);  /* *pcbErrorMsg */
```

Function arguments:

Table 84. SQLGetDiagRec arguments

Data type	Argument	Use	Description
SQLSMALLINT	*HandleType*	input	A handle type identifier that describes the type of handle for which diagnostics are desired. Must be one of the following: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
SQLHANDLE	*Handle*	input	A handle for the diagnostic data structure, of the type indicated by *HandleType*.
SQLSMALLINT	*RecNumber*	input	Indicates the status record from which the application seeks information. Status records are numbered from 1.
SQLCHAR *	*SQLState*	output	Pointer to a buffer in which to return 5 characters plus a NULL terminator for the SQLSTATE code pertaining to the diagnostic record *RecNumber*. The first two characters indicate the class; the next three indicate the subclass.
SQLINTEGER *	*NativeErrorPtr*	output	Pointer to a buffer in which to return the native error code, specific to the data source.
SQLCHAR *	*MessageText*	output	Pointer to a buffer in which to return the error message text. The fields returned by `SQLGetDiagRec()` are contained in a text string.
SQLINTEGER	*BufferLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *MessageText* buffer.
SQLSMALLINT *	*TextLengthPtr*	output	Pointer to a buffer in which to return the total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in **MessageText*. If the number of SQLCHAR or SQLWCHAR elements available to return is greater than *BufferLength*, then the error message text in **MessageText* is truncated to *BufferLength* minus the length of a null-termination character.

Usage:

An application typically calls `SQLGetDiagRec()` when a previous call to a DB2 CLI function has returned anything other than SQL_SUCCESS. However, any function can post zero or more errors each time it is called, so an application can call `SQLGetDiagRec()` after any function call. An application can call `SQLGetDiagRec()` multiple times to return some or all of the records in the diagnostic data structure.

`SQLGetDiagRec()` returns a character string containing the following fields of the diagnostic data structure record:

SQL_DIAG_MESSAGE_TEXT (return type CHAR *)
An informational message on the error or warning.

SQL_DIAG_NATIVE (return type SQLINTEGER)
A driver/data-source-specific native error code. If there is no native error code, the driver returns 0.

SQL_DIAG_SQLSTATE (return type CHAR *)
A five-character SQLSTATE diagnostic code.

`SQLGetDiagRec()` cannot be used to return fields from the header of the diagnostic data structure (the *RecNumber* argument must be greater than 0). The application should call `SQLGetDiagField()` for this purpose.

`SQLGetDiagRec()` retrieves only the diagnostic information most recently associated with the handle specified in the *Handle* argument. If the application calls another function, except `SQLGetDiagRec()` or `SQLGetDiagField()`, any diagnostic information from the previous calls on the same handle is lost.

An application can scan all diagnostic records by looping, incrementing *RecNumber*, as long as `SQLGetDiagRec()` returns SQL_SUCCESS. Calls to `SQLGetDiagRec()` are non-destructive to the header and record fields. The application can call `SQLGetDiagRec()` again at a later time to retrieve a field from a record, as long as no other function, except `SQLGetDiagRec()` or `SQLGetDiagField()`, has been called in the interim. The application can call `SQLGetDiagField()` to retrieve the value of the SQL_DIAG_NUMBER field, which is the total number of diagnostic records available. `SQLGetDiagRec()` should then be called that many times.

HandleType argument

Each handle type can have diagnostic information associated with it. The *HandleType* argument denotes the handle type of *Handle*.

Some header and record fields cannot be returned for all types of handles: environment, connection, statement, and descriptor. Those handles for which a field is not applicable are indicated in the list of header and record fields for the *DiagIdentifier* argument.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

`SQLGetDiagRec()` does not post error values for itself. It uses the following return values to report the outcome of its own execution:
- SQL_SUCCESS: The function successfully returned diagnostic information.
- SQL_SUCCESS_WITH_INFO: The **MessageText* buffer was too small to hold the requested diagnostic message. No diagnostic records were generated. To determine that a truncation occurred, the application must compare *BufferLength* to the actual number of bytes available, which is written to **StringLengthPtr*.
- SQL_INVALID_HANDLE: The handle indicated by *HandleType* and *Handle* was not a valid handle.
- SQL_ERROR: One of the following occurred:

  - *RecNumber* was negative or 0.
  - *BufferLength* was less than zero.
- SQL_NO_DATA: *RecNumber* was greater than the number of diagnostic records that existed for the handle specified in *Handle*. The function also returns SQL_NO_DATA for any positive *RecNumber* if there are no diagnostic records for *Handle*.

Example:

```
/* get multiple fields settings of diagnostic record */
SQLGetDiagRec(SQL_HANDLE_STMT,
              hstmt,
              1,
              sqlstate,
              &sqlcode,
              message,
              200,
              &length);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Header and record fields for the DiagIdentifier argument (CLI)" on page 385
- "SQLGetDiagField function (CLI) - Get a field of diagnostic data" on page 168
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spclient.c -- Call various stored procedures"
- "utilcli.c -- Utility functions used by DB2 CLI samples"

SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value

Purpose:

Specification:	DB2 CLI 2.1		ISO CLI

`SQLGetEnvAttr()` returns the current setting for the specified environment attribute.

These options are set using the `SQLSetEnvAttr()` function.

Syntax:

```
SQLRETURN   SQLGetEnvAttr    (
               SQLHENV           EnvironmentHandle,  /* henv */
               SQLINTEGER        Attribute,
               SQLPOINTER        ValuePtr,           /* Value */
               SQLINTEGER        BufferLength,
               SQLINTEGER        *StringLengthPtr);  /* StringLength */
```

Function arguments:

Table 85. SQLGetEnvAttr arguments

Data type	Argument	Use	Description
SQLHENV	*EnvironmentHandle*	input	Environment handle.
SQLINTEGER	*Attribute*	input	Attribute to receive. Refer to the list of environment attributes and their descriptions.
SQLPOINTER	*ValuePtr*	output	A pointer to memory in which to return the current value of the attribute specified by *Attribute*.
SQLINTEGER	*BufferLength*	input	Maximum size of buffer pointed to by *ValuePtr*, if the attribute value is a character string; otherwise, ignored.
SQLINTEGER *	*StringLengthPtr*	output	Pointer to a buffer in which to return the total number of bytes (excluding the number of bytes returned for the null-termination character) available to return in *ValuePtr*. If *ValuePtr* is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than or equal to *BufferLength*, the data in *ValuePtr* is truncated to *BufferLength* minus the length of a null-termination character and is null-terminated by DB2 CLI.

If *Attribute* does not denote a string, then DB2 CLI ignores *BufferLength* and does not set *StringLengthPtr*.

Usage:

`SQLGetEnvAttr()` can be called at any time between the allocation and freeing of the environment handle. It obtains the current value of the environment attribute.

Return codes:
- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 86. SQLGetEnvAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY092	Option type out of range.	An invalid *Attribute* value was specified.

Restrictions:

None.

Example:

```
/* retrieve the current environment attribute value */
cliRC = SQLGetEnvAttr(henv, SQL_ATTR_OUTPUT_NTS, &output_nts, 0, NULL);
```

Related concepts:

- "Handle freeing in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Initializing CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "Environment attributes (CLI) list" on page 321
- "SQLSetEnvAttr function (CLI) - Set environment attribute" on page 284

Related samples:

- "cli_info.c -- How to get and set environment attributes at the client level"

SQLGetFunctions function (CLI) - Get functions

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

`SQLGetFunctions()` can be used to query whether a specific DB2 CLI or ODBC function is supported. This allows applications to adapt to varying levels of support when connecting to different database servers.

A connection to a database server must exist before calling this function.

Syntax:

```
SQLRETURN   SQLGetFunctions  (
               SQLHDBC          ConnectionHandle,  /* hdbc */
               SQLUSMALLINT     FunctionId,        /* fFunction */
               SQLUSMALLINT     *SupportedPtr);    /* pfExists */
```

Function arguments:

Table 87. SQLGetFunctions arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Database connection handle.
SQLUSMALLINT	*FunctionId*	input	The function being queried.
SQLUSMALLINT *	*SupportedPtr*	output	Pointer to location where this function will return SQL_TRUE or SQL_FALSE depending on whether the function being queried is supported.

Usage:

If *FunctionId* is set to SQL_API_ALL_FUNCTIONS, then *SupportedPtr* must point to an SQLSMALLINT array of 100 elements. The array is indexed by the *FunctionId* values used to identify many of the functions. Some elements of the array are unused and reserved. Since some *FunctionId* values are greater than 100, the array

method can not be used to obtain a list of functions. The SQLGetFunctions() call must be explicitly issued for all *FunctionId* values equal to or above 100. The complete set of *FunctionId* values is defined in sqlcli1.h.

Note: The LOB support functions (SQLGetLength(), SQLGetPosition(), SQLGetSubString(), SQLBindFileToCol(), SQLBindFileToCol()) are not supported when connected to IBM RDBMSs that do not support LOB data types.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 88. SQLGetFunctions SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	SQLGetFunctions() was called before a database connection was established.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Authorization:

None.

Example:

```
/* check to see if SQLGetInfo() is supported */
cliRC = SQLGetFunctions(hdbc, SQL_API_SQLGETINFO, &supported);
```

References:

None.

Related concepts:
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:
- "dbinfo.c -- How to get and set information at the database level"
- "dbinfo.out -- HOW TO GET AND SET DATABASE INFORMATION (CLI)"

- "ilinfo.c -- How to get information at the installation image level"
- "ininfo.c -- How to get information at the instance level"

SQLGetInfo function (CLI) - Get general information

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLGetInfo()` returns general information about the DBMS that the application is currently connected to.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLGetInfoW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetInfo       (
               SQLHDBC           ConnectionHandle,   /* hdbc */
               SQLUSMALLINT      InfoType,           /* fInfoType */
               SQLPOINTER        InfoValuePtr,       /* rgbInfoValue */
               SQLSMALLINT       BufferLength,       /* cbInfoValueMax */
               SQLSMALLINT       *StringLengthPtr);  /* pcbInfoValue */
```

Function arguments:

Table 89. SQLGetInfo arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Database connection handle
SQLUSMALLINT	*InfoType*	input	The type of information desired. The possible values for this argument are described in "Information returned by SQLGetInfo()" on page 182.
SQLPOINTER	*InfoValuePtr*	output (also input)	Pointer to buffer where this function will store the desired information. Depending on the type of information being retrieved, 5 types of information can be returned: • 16 bit integer value • 32 bit integer value • 32 bit binary value • 32 bit mask • null-terminated character string If the *InfoType* argument is SQL_DRIVER_HDESC or SQL_DRIVER_HSTMT, the *InfoValuePtr* argument is both input and output.
SQLSMALLINT	*BufferLength*	input	Maximum length of the buffer pointed by *InfoValuePtr* pointer. If **InfoValuePtr* is a Unicode string, the *BufferLength* argument must be an even number.

Table 89. SQLGetInfo arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	*StringLengthPtr*	output	Pointer to location where this function will return the total number of bytes of information available to return. In the case of string output, this size does not include the null terminating character. If the value in the location pointed to by *StringLengthPtr* is greater than the size of the *InfoValuePtr* buffer as specified in *BufferLength*, then the string output information would be truncated to *BufferLength* - 1 bytes and the function would return with SQL_SUCCESS_WITH_INFO.

Usage:

Refer to "Information returned by SQLGetInfo()" on page 182 for a list of the possible values of *InfoType* and a description of the information that `SQLGetInfo()` would return for that value.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 90. SQLGetInfo SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The requested information was returned as a string and its length exceeded the length of the application buffer as specified in *BufferLength*. The argument *StringLengthPtr* contains the actual (not truncated) length of the requested information. (Function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed.	The type of information requested in *InfoType* requires an open connection. Only SQL_ODBC_VER does not require an open connection.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY090	Invalid string or buffer length.	The value specified for argument *BufferLength* was less than 0.
HY096	Information type out of range.	An invalid *InfoType* was specified.
HYC00	Driver not capable.	The value specified in the argument *InfoType* is not supported by either DB2 CLI or the data source.

Restrictions:

None.

Example:

```
/* get server name information */
cliRC = SQLGetInfo(hdbc, SQL_DBMS_NAME, imageInfoBuf, 255, &outlen);

/* ... */

/* get client driver name information */
cliRC = SQLGetInfo(hdbc, SQL_DRIVER_NAME, imageInfoBuf, 255, &outlen);
```

Information returned by SQLGetInfo():

Note: DB2 CLI returns a value for each *InfoType* in this table. If the *InfoType* does not apply or is not supported, the result is dependent on the return type. If the return type is a:
- Character string ("Y" or "N"), "N" is returned.
- Character string (not "Y" or "N"), an empty string is returned.
- 32-bit integer, 0 (zero) is returned.
- 32-bit mask, 0 (zero) is returned.

SQL_ACCESSIBLE_PROCEDURES (string)
A character string of "Y" indicates that the user can execute all procedures returned by the function `SQLProcedures()`. "N" indicates there might be procedures returned that the user cannot execute.

SQL_ACCESSIBLE_TABLES (string)
A character string of "Y" indicates that the user is guaranteed SELECT privilege to all tables returned by the function `SQLTables()`. "N" indicates that there might be tables returned that the user cannot access.

SQL_AGGREGATE_FUNCTIONS (32-bit mask)
A bitmask enumerating support for aggregation functions:
- SQL_AF_ALL
- SQL_AF_AVG
- SQL_AF_COUNT
- SQL_AF_DISTINCT
- SQL_AF_MAX
- SQL_AF_MIN
- SQL_AF_SUM

SQL_ALTER_DOMAIN (32-bit mask)
DB2 CLI returns 0 indicating that the ALTER DOMAIN statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_AD_ADD_CONSTRAINT_DEFERRABLE
- SQL_AD_ADD_CONSTRAINT_NON_DEFERRABLE
- SQL_AD_ADD_CONSTRAINT_INITIALLY_DEFERRED
- SQL_AD_ADD_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_AD_ADD_DOMAIN_CONSTRAINT
- SQL_AD_ADD_DOMAIN_DEFAULT
- SQL_AD_CONSTRAINT_NAME_DEFINITION
- SQL_AD_DROP_DOMAIN_CONSTRAINT
- SQL_AD_DROP_DOMAIN_DEFAULT

SQL_ALTER_TABLE (32-bit mask)
Indicates which clauses in the ALTER TABLE statement are supported by the DBMS.
- SQL_AT_ADD_COLUMN_COLLATION

- SQL_AT_ADD_COLUMN_DEFAULT
- SQL_AT_ADD_COLUMN_SINGLE
- SQL_AT_ADD_CONSTRAINT
- SQL_AT_ADD_TABLE_CONSTRAINT
- SQL_AT_CONSTRAINT_NAME_DEFINITION
- SQL_AT_DROP_COLUMN_CASCADE
- SQL_AT_DROP_COLUMN_DEFAULT
- SQL_AT_DROP_COLUMN_RESTRICT
- SQL_AT_DROP_TABLE_CONSTRAINT_CASCADE
- SQL_AT_DROP_TABLE_CONSTRAINT_RESTRICT
- SQL_AT_SET_COLUMN_DEFAULT
- SQL_AT_CONSTRAINT_INITIALLY_DEFERRED
- SQL_AT_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_AT_CONSTRAINT_DEFERRABLE
- SQL_AT_CONSTRAINT_NON_DEFERRABLE

SQL_APPLICATION_CODEPAGE (32-bit unsigned integer)
Indicates the application code page.

SQL_ASYNC_MODE (32-bit unsigned integer)
Indicates the level of asynchronous support in the driver:
- SQL_AM_CONNECTION, connection level asynchronous execution is supported. Either all statement handles associated with a given connection handle are in asynchronous mode, or all are in synchronous mode. A statement handle on a connection cannot be in asynchronous mode while another statement handle on the same connection is in synchronous mode, and vice versa.
- SQL_AM_STATEMENT, statement level asynchronous execution is supported. Some statement handles associated with a connection handle can be in asynchronous mode, while other statement handles on the same connection are in synchronous mode.
- SQL_AM_NONE, asynchronous mode is not supported.

  This value is also returned if the DB2 CLI/ODBC configuration keyword ASYNCENABLE is set to disable asynchronous execution.

SQL_BATCH_ROW_COUNT (32-bit mask)
Indicates how row counts are dealt with. DB2 CLI always returns SQL_BRC_ROLLED_UP indicating that row counts for consecutive INSERT, DELETE, or UPDATE statements are rolled up into one.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_BRC_PROCEDURES
- SQL_BRC_EXPLICIT

SQL_BATCH_SUPPORT (32-bit mask)
Indicates which levels of batches are supported:
- SQL_BS_SELECT_EXPLICIT, supports explicit batches that can have result-set generating statements.
- SQL_BS_ROW_COUNT_EXPLICIT, supports explicit batches that can have row-count generating statements.
- SQL_BS_SELECT_PROC, supports explicit procedures that can have result-set generating statements.
- SQL_BS_ROW_COUNT_PROC, supports explicit procedures that can have row-count generating statements.

SQL_BOOKMARK_PERSISTENCE (32-bit mask)
Indicates when bookmarks remain valid after an operation:

- SQL_BP_CLOSE, bookmarks are valid after an application calls `SQLFreeStmt()` with the SQL_CLOSE option, or `SQLCloseCursor()` to close the cursor associated with a statement.
- SQL_BP_DELETE, the bookmark for a row is valid after that row has been deleted.
- SQL_BP_DROP, bookmarks are valid after an application calls `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_STMT to drop a statement.
- SQL_BP_TRANSACTION, bookmarks are valid after an application commits or rolls back a transaction.
- SQL_BP_UPDATE, the bookmark for a row is valid after any column in that row has been updated, including key columns.
- SQL_BP_OTHER_HSTMT, a bookmark associated with one statement can be used with another statement. Unless SQL_BP_CLOSE or SQL_BP_DROP is specified, the cursor on the first statement must be open.

SQL_CATALOG_LOCATION (16-bit integer)
A 16-bit integer value indicated the position of the qualifier in a qualified table name. DB2 CLI always returns SQL_CL_START for this information type. ODBC also defines the value SQL_CL_END which is not returned by DB2 CLI.

In previous versions of DB2 CLI this *InfoType* was SQL_QUALIFIER_LOCATION.

SQL_CATALOG_NAME (string)
A character string of ″Y″ indicates that the server supports catalog names. ″N″ indicates that catalog names are not supported.

SQL_CATALOG_NAME_SEPARATOR (string)
The character(s) used as a separator between a catalog name and the qualified name element that follows or precedes it.

In previous versions of DB2 CLI this *InfoType* was SQL_QUALIFIER_NAME_SEPARATOR.

SQL_CATALOG_TERM (string)
The database vendor's terminology for a qualifier (catalog).

The name that the vendor uses for the high order part of a three part name.

If the target DBMS does not support 3-part naming, a zero-length string is returned.

In previous versions of DB2 CLI this *InfoType* was SQL_QUALIFIER_TERM.

SQL_CATALOG_USAGE (32-bit mask)
This is similar to SQL_SCHEMA_USAGE except that this is used for catalogs.

A 32-bit mask enumerating the statements in which catalogs can be used:

- SQL_CU_DML_STATEMENTS - Catalogs are supported in all DML statements.
- SQL_CU_INDEX_DEFINITION - Catalogs are supported in all index definition statements.
- SQL_CU_PRIVILEGE_DEFINITION - Catalogs are supported in all privilege definition statements.

- SQL_CU_PROCEDURE_INVOCATION - Catalogs are supported in the ODBC procedure invocation statement.
- SQL_CU_TABLE_DEFINITION - Catalogs are supported in all table definition statements.

A value of zero is returned if catalogs are not supported by the data source.

In previous versions of DB2 CLI, this *InfoType* was SQL_QUALIFIER_USAGE.

SQL_COLLATION_SEQ (string)
The name of the collation sequence. This is a character string that indicates the name of the default collation for the default character set for this server (for example ISO 8859-1 or EBCDIC). If this is unknown, an empty string will be returned.

SQL_COLUMN_ALIAS (string)
Returns ″Y″ if column aliases are supported, or ″N″ if they are not.

SQL_CONCAT_NULL_BEHAVIOR (16-bit integer)
Indicates how the concatenation of NULL valued character data type columns with non-NULL valued character data type columns is handled.
- SQL_CB_NULL - indicates the result is a NULL value (this is the case for IBM RDBMS).
- SQL_CB_NON_NULL - indicates the result is a concatenation of non-NULL column values.

SQL_CONVERT_BIGINT
SQL_CONVERT_BINARY
SQL_CONVERT_BIT
SQL_CONVERT_CHAR
SQL_CONVERT_DATE
SQL_CONVERT_DECIMAL
SQL_CONVERT_DOUBLE
SQL_CONVERT_FLOAT
SQL_CONVERT_INTEGER
SQL_CONVERT_INTERVAL_YEAR_MONTH
SQL_CONVERT_INTERVAL_DAY_TIME
SQL_CONVERT_LONGVARBINARY
SQL_CONVERT_LONGVARCHAR
SQL_CONVERT_NUMERIC
SQL_CONVERT_REAL
SQL_CONVERT_SMALLINT
SQL_CONVERT_TIME
SQL_CONVERT_TIMESTAMP
SQL_CONVERT_TINYINT
SQL_CONVERT_VARBINARY
SQL_CONVERT_VARCHAR
SQL_CONVERT_WCHAR
SQL_CONVERT_WLONGVARCHAR
SQL_CONVERT_WVARCHAR

(all above are 32-bit masks)
Indicates the conversions supported by the data source with the CONVERT scalar function for data of the type named in the *InfoType*. If the bitmask equals zero, the data source does not support any conversions for the data of the named type, including conversions to the same data type.

For example, to find out if a data source supports the conversion of SQL_INTEGER data to the SQL_DECIMAL data type, an application calls `SQLGetInfo()` with *InfoType* of SQL_CONVERT_INTEGER. The application then ANDs the returned bitmask with SQL_CVT_DECIMAL. If the resulting value is nonzero then the conversion is supported.

The following bitmasks are used to determine which conversions are supported:

- SQL_CVT_BIGINT
- SQL_CVT_BINARY
- SQL_CVT_BIT
- SQL_CVT_CHAR
- SQL_CVT_DATE
- SQL_CVT_DECIMAL
- SQL_CVT_DOUBLE
- SQL_CVT_FLOAT
- SQL_CVT_INTEGER
- SQL_CVT_INTERVAL_YEAR_MONTH
- SQL_CVT_INTERVAL_DAY_TIME
- SQL_CVT_LONGVARBINARY
- SQL_CVT_LONGVARCHAR
- SQL_CVT_NUMERIC
- SQL_CVT_REAL
- SQL_CVT_SMALLINT
- SQL_CVT_TIME
- SQL_CVT_TIMESTAMP
- SQL_CVT_TINYINT
- SQL_CVT_VARBINARY
- SQL_CVT_VARCHAR
- SQL_CVT_WCHAR
- SQL_CVT_WLONGVARCHAR
- SQL_CVT_WVARCHAR

SQL_CONNECT_CODEPAGE (32-bit unsigned integer)
Indicates the code page of the current connection.

SQL_CONVERT_FUNCTIONS (32-bit mask)
Indicates the scalar conversion functions supported by the driver and associated data source.

DB2 CLI Version 2.1.1 and later supports ODBC scalar conversions between char variables (CHAR, VARCHAR, LONG VARCHAR and CLOB) and DOUBLE (or FLOAT).

- SQL_FN_CVT_CONVERT - used to determine which conversion functions are supported.

SQL_CORRELATION_NAME (16-bit integer)
Indicates the degree of correlation name support by the server:

- SQL_CN_ANY, supported and can be any valid user-defined name.
- SQL_CN_NONE, correlation name not supported.
- SQL_CN_DIFFERENT, correlation name supported but it must be different than the name of the table that it represent.

SQL_CREATE_ASSERTION (32-bit mask)
Indicates which clauses in the CREATE ASSERTION statement are supported by the DBMS. DB2 CLI always returns zero; the CREATE ASSERTION statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:

- SQL_CA_CREATE_ASSERTION
- SQL_CA_CONSTRAINT_INITIALLY_DEFERRED
- SQL_CA_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_CA_CONSTRAINT_DEFERRABLE
- SQL_CA_CONSTRAINT_NON_DEFERRABLE

SQL_CREATE_CHARACTER_SET (32-bit mask)
Indicates which clauses in the CREATE CHARACTER SET statement are supported by the DBMS. DB2 CLI always returns zero; the CREATE CHARACTER SET statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_CCS_CREATE_CHARACTER_SET
- SQL_CCS_COLLATE_CLAUSE
- SQL_CCS_LIMITED_COLLATION

SQL_CREATE_COLLATION (32-bit mask)
Indicates which clauses in the CREATE COLLATION statement are supported by the DBMS. DB2 CLI always returns zero; the CREATE COLLATION statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_CCOL_CREATE_COLLATION

SQL_CREATE_DOMAIN (32-bit mask)
Indicates which clauses in the CREATE DOMAIN statement are supported by the DBMS. DB2 CLI always returns zero; the CREATE DOMAIN statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_CDO_CREATE_DOMAIN
- SQL_CDO_CONSTRAINT_NAME_DEFINITION
- SQL_CDO_DEFAULT
- SQL_CDO_CONSTRAINT
- SQL_CDO_COLLATION
- SQL_CDO_CONSTRAINT_INITIALLY_DEFERRED
- SQL_CDO_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_CDO_CONSTRAINT_DEFERRABLE
- SQL_CDO_CONSTRAINT_NON_DEFERRABLE

SQL_CREATE_SCHEMA (32-bit mask)
Indicates which clauses in the CREATE SCHEMA statement are supported by the DBMS:
- SQL_CS_CREATE_SCHEMA
- SQL_CS_AUTHORIZATION
- SQL_CS_DEFAULT_CHARACTER_SET

SQL_CREATE_TABLE (32-bit mask)
Indicates which clauses in the CREATE TABLE statement are supported by the DBMS.

The following bitmasks are used to determine which clauses are supported:
- SQL_CT_CREATE_TABLE
- SQL_CT_TABLE_CONSTRAINT
- SQL_CT_CONSTRAINT_NAME_DEFINITION

The following bits specify the ability to create temporary tables:
- SQL_CT_COMMIT_PRESERVE, deleted rows are preserved on commit.
- SQL_CT_COMMIT_DELETE, deleted rows are deleted on commit.

- SQL_CT_GLOBAL_TEMPORARY, global temporary tables can be created.
- SQL_CT_LOCAL_TEMPORARY, local temporary tables can be created.

The following bits specify the ability to create column constraints:
- SQL_CT_COLUMN_CONSTRAINT, specifying column constraints is supported.
- SQL_CT_COLUMN_DEFAULT, specifying column defaults is supported.
- SQL_CT_COLUMN_COLLATION, specifying column collation is supported.

The following bits specify the supported constraint attributes if specifying column or table constraints is supported:
- SQL_CT_CONSTRAINT_INITIALLY_DEFERRED
- SQL_CT_CONSTRAINT_INITIALLY_IMMEDIATE
- SQL_CT_CONSTRAINT_DEFERRABLE
- SQL_CT_CONSTRAINT_NON_DEFERRABLE

SQL_CREATE_TRANSLATION (32-bit mask)
Indicates which clauses in the CREATE TRANSLATION statement are supported by the DBMS. DB2 CLI always returns zero; the CREATE TRANSLATION statement is not supported.

ODBC also defines the following value that is not returned by DB2 CLI:
- SQL_CTR_CREATE_TRANSLATION

SQL_CREATE_VIEW (32-bit mask)
Indicates which clauses in the CREATE VIEW statement are supported by the DBMS:
- SQL_CV_CREATE_VIEW
- SQL_CV_CHECK_OPTION
- SQL_CV_CASCADED
- SQL_CV_LOCAL

A return value of 0 means that the CREATE VIEW statement is not supported.

SQL_CURSOR_COMMIT_BEHAVIOR (16-bit integer)
Indicates how a COMMIT operation affects cursors. A value of:
- SQL_CB_DELETE, destroy cursors and drops access plans for dynamic SQL statements.
- SQL_CB_CLOSE, destroy cursors, but retains access plans for dynamic SQL statements (including non-query statements)
- SQL_CB_PRESERVE, retains cursors and access plans for dynamic statements (including non-query statements). Applications can continue to fetch data, or close the cursor and re-execute the query without re-preparing the statement.

  Note: After COMMIT, a FETCH must be issued to reposition the cursor before actions such as positioned updates or deletes can be taken.

SQL_CURSOR_ROLLBACK_BEHAVIOR (16-bit integer)
Indicates how a ROLLBACK operation affects cursors. A value of:
- SQL_CB_DELETE, destroy cursors and drops access plans for dynamic SQL statements.
- SQL_CB_CLOSE, destroy cursors, but retains access plans for dynamic SQL statements (including non-query statements)

- SQL_CB_PRESERVE, retains cursors and access plans for dynamic statements (including non-query statements). Applications can continue to fetch data, or close the cursor and re-execute the query without re-preparing the statement.

  Note: DB2 servers do not have the SQL_CB_PRESERVE property.

SQL_CURSOR_SENSITIVITY (32-bit unsigned integer)
Indicates support for cursor sensitivity:

- SQL_INSENSITIVE, all cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor within the same transaction.
- SQL_UNSPECIFIED, it is unspecified whether cursors on the statement handle make visible the changes made to a result set by another cursor within the same transaction. Cursors on the statement handle might make visible none, some, or all such changes.
- SQL_SENSITIVE, cursors are sensitive to changes made by other cursors within the same transaction.

SQL_DATA_SOURCE_NAME (string)
A character string with the data source name used during connection. If the application called `SQLConnect()`, this is the value of the *szDSN* argument. If the application called `SQLDriverConnect()` or `SQLBrowseConnect()`, this is the value of the DSN keyword in the connection string passed to the driver. If the connection string did not contain the DSN keyword, this is an empty string.

SQL_DATA_SOURCE_READ_ONLY (string)
A character string of ″Y″ indicates that the database is set to READ ONLY mode, ″N″ indicates that is not set to READ ONLY mode. This characteristic pertains only to the data source itself; it is not characteristic of the driver that enables access to the data source.

SQL_DATABASE_CODEPAGE (32-bit unsigned integer)
Indicates the code page of the database that the application is currently connected to.

SQL_DATABASE_NAME (string)
The name of the current database in use

Note: This string is the same as that returned by the SELECT CURRENT SERVER statement on non-host systems. For host databases, such as DB2 for OS/390 or DB2 for OS/400®, the string returned is the DCS database name that was provided when the CATALOG DCS DATABASE DIRECTORY command was issued at the DB2 Connect™ gateway.

SQL_DATETIME_LITERALS (32-bit unsigned integer)
Indicates the datetime literals that are supported by the DBMS. DB2 CLI always returns zero; datetime literals are not supported.

ODBC also defines the following values that are not returned by DB2 CLI:

- SQL_DL_SQL92_DATE
- SQL_DL_SQL92_TIME
- SQL_DL_SQL92_TIMESTAMP
- SQL_DL_SQL92_INTERVAL_YEAR
- SQL_DL_SQL92_INTERVAL_MONTH
- SQL_DL_SQL92_INTERVAL_DAY
- SQL_DL_SQL92_INTERVAL_HOUR

- SQL_DL_SQL92_INTERVAL_MINUTE
- SQL_DL_SQL92_INTERVAL_SECOND
- SQL_DL_SQL92_INTERVAL_YEAR_TO_MONTH
- SQL_DL_SQL92_INTERVAL_DAY_TO_HOUR
- SQL_DL_SQL92_INTERVAL_DAY_TO_MINUTE
- SQL_DL_SQL92_INTERVAL_DAY_TO_SECOND
- SQL_DL_SQL92_INTERVAL_HOUR_TO_MINUTE
- SQL_DL_SQL92_INTERVAL_HOUR_TO_SECOND
- SQL_DL_SQL92_INTERVAL_MINUTE_TO_SECOND

SQL_DBMS_NAME (string)
The name of the DBMS product being accessed

For example:
- ″DB2/6000″
- ″DB2/2″

SQL_DBMS_VER (string)
The Version of the DBMS product accessed. A string of the form 'mm.vv.rrrr' where mm is the major version, vv is the minor version and rrrr is the release. For example, ″0r.01.0000″ translates to major version r, minor version 1, release 0.

SQL_DDL_INDEX (32-bit unsigned integer)
Indicates support for the creation and dropping of indexes:
- SQL_DI_CREATE_INDEX
- SQL_DI_DROP_INDEX

SQL_DEFAULT_TXN_ISOLATION (32-bit mask)
The default transaction isolation level supported

One of the following masks are returned:

- SQL_TXN_READ_UNCOMMITTED = Changes are immediately perceived by all transactions (dirty read, non-repeatable read, and phantoms are possible).

  This is equivalent to IBM's Uncommitted Read level.
- SQL_TXN_READ_COMMITTED = Row read by transaction 1 can be altered and committed by transaction 2 (non-repeatable read and phantoms are possible)

  This is equivalent to IBM's Cursor Stability level.
- SQL_TXN_REPEATABLE_READ = A transaction can add or remove rows matching the search condition or a pending transaction (repeatable read, but phantoms are possible)

  This is equivalent to IBM's Read Stability level.
- SQL_TXN_SERIALIZABLE = Data affected by pending transaction is not available to other transactions (repeatable read, phantoms are not possible)

  This is equivalent to IBM's Repeatable Read level.
- SQL_TXN_VERSIONING = Not applicable to IBM DBMSs.
- SQL_TXN_NOCOMMIT = Any changes are effectively committed at the end of a successful operation; no explicit commit or rollback is allowed.

  This is a DB2 Universal Database for AS/400 (DB2 UDB for AS/400) isolation level.

In IBM terminology,
- SQL_TXN_READ_UNCOMMITTED is Uncommitted Read;

- SQL_TXN_READ_COMMITTED is Cursor Stability;
- SQL_TXN_REPEATABLE_READ is Read Stability;
- SQL_TXN_SERIALIZABLE is Repeatable Read.

SQL_DESCRIBE_PARAMETER (string)
″Y″ if parameters can be described; ″N″ if not.

SQL_DM_VER (string)
Reserved.

SQL_DRIVER_HDBC (32 bits)
DB2 CLI's database handle

SQL_DRIVER_HDESC (32 bits)
DB2 CLI's descriptor handle

SQL_DRIVER_HENV (32 bits)
DB2 CLI's environment handle

SQL_DRIVER_HLIB (32 bits)
Reserved.

SQL_DRIVER_HSTMT (32 bits)
DB2 CLI's statement handle

In an ODBC environment with an ODBC Driver Manager, if *InfoType* is set to SQL_DRIVER_HSTMT, the Driver Manager statement handle (the one returned from `SQLAllocStmt()`) must be passed on input in *rgbInfoValue* from the application. In this case *rgbInfoValue* is both an input and an output argument. The ODBC Driver Manager is responsible for returning the mapped value. ODBC applications wishing to call DB2 CLI specific functions (such as the LOB functions) can access them, by passing these handle values to the functions after loading the DB2 CLI library and issuing an operating system call to invoke the desired functions.

SQL_DRIVER_NAME (string)
The file name of the DB2 CLI implementation.

SQL_DRIVER_ODBC_VER (string)
The version number of ODBC that the Driver supports. DB2 CLI will return ″03.00″.

SQL_DRIVER_VER (string)
The version of the CLI driver. A string of the form 'mm.vv.rrrr' where mm is the major version, vv is the minor version and rrrr is the release. For example, ″05.01.0000″ translates to major version 5, minor version 1, release 0.

SQL_DROP_ASSERTION (32-bit unsigned integer)
Indicates which clause in the DROP ASSERTION statement is supported by the DBMS. DB2 CLI always returns zero; the DROP ASSERTION statement is not supported.

ODBC also defines the following value that is not returned by DB2 CLI:
- SQL_DA_DROP_ASSERTION

SQL_DROP_CHARACTER_SET (32-bit unsigned integer)
Indicates which clause in the DROP CHARACTER SET statement is supported by the DBMS. DB2 CLI always returns zero; the DROP CHARACTER SET statement is not supported.

ODBC also defines the following value that is not returned by DB2 CLI:
- SQL_DCS_DROP_CHARACTER_SET

SQL_DROP_COLLATION (32-bit unsigned integer)
Indicates which clause in the DROP COLLATION statement is supported by the DBMS. DB2 CLI always returns zero; the DROP COLLATION statement is not supported.

ODBC also defines the following value that is not returned by DB2 CLI:
- SQL_DC_DROP_COLLATION

SQL_DROP_DOMAIN (32-bit unsigned integer)
Indicates which clauses in the DROP DOMAIN statement are supported by the DBMS. DB2 CLI always returns zero; the DROP DOMAIN statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_DD_DROP_DOMAIN
- SQL_DD_CASCADE
- SQL_DD_RESTRICT

SQL_DROP_SCHEMA (32-bit unsigned integer)
Indicates which clauses in the DROP SCHEMA statement are supported by the DBMS. DB2 CLI always returns zero; the DROP SCHEMA statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_DS_CASCADE
- SQL_DS_RESTRICT

SQL_DROP_TABLE (32-bit unsigned integer)
Indicates which clauses in the DROP TABLE statement are supported by the DBMS:
- SQL_DT_DROP_TABLE
- SQL_DT_CASCADE
- SQL_DT_RESTRICT

SQL_DROP_TRANSLATION (32-bit unsigned integer)
Indicates which clauses in the DROP TRANSLATION statement are supported by the DBMS. DB2 CLI always returns zero; the DROP TRANSLATION statement is not supported.

ODBC also defines the following value that is not returned by DB2 CLI:
- SQL_DTR_DROP_TRANSLATION

SQL_DROP_VIEW (32-bit unsigned integer)
Indicates which clauses in the DROP VIEW statement are supported by the DBMS. DB2 CLI always returns zero; the DROP VIEW statement is not supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_DV_CASCADE
- SQL_DV_RESTRICT

SQL_DTC_TRANSITION_COST (32-bit unsigned mask)
Used by Microsoft® Transaction Server to determine whether or not the enlistment process for a connection is expensive. DB2 CLI returns:
- SQL_DTC_ENLIST_EXPENSIVE
- SQL_DTC_UNENLIST_EXPENSIVE

SQL_DYNAMIC_CURSOR_ATTRIBUTES1 (32-bit mask)
Indicates the attributes of a dynamic cursor that are supported by DB2 CLI (subset 1 of 2).
- SQL_CA1_NEXT
- SQL_CA1_ABSOLUTE

- SQL_CA1_RELATIVE
- SQL_CA1_BOOKMARK
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_DYNAMIC_CURSOR_ATTRIBUTES2 (32-bit mask)
Indicates the attributes of a dynamic cursor that are supported by DB2 CLI (subset 2 of 2).

- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_OPT_ROWVER_CONCURRENCY
- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_EXPRESSIONS_IN_ORDERBY (string)
The character string ″Y″ indicates the database server supports the DIRECT specification of expressions in the ORDER BY list, ″N″ indicates that it does not.

SQL_FETCH_DIRECTION (32-bit mask)
The supported fetch directions.

The following bit-masks are used in conjunction with the flag to determine which options are supported.

- SQL_FD_FETCH_NEXT
- SQL_FD_FETCH_FIRST
- SQL_FD_FETCH_LAST
- SQL_FD_FETCH_PREV
- SQL_FD_FETCH_ABSOLUTE
- SQL_FD_FETCH_RELATIVE
- SQL_FD_FETCH_RESUME

SQL_FILE_USAGE (16-bit integer)
Indicates how a single-tier driver directly treats files in a data source. The DB2 CLI driver is not a single-tier driver and therefor always returns SQL_FILE_NOT_SUPPORTED.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_FILE_TABLE
- SQL_FILE_CATALOG

SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1 (32-bit mask)
Indicates the attributes of a forward-only cursor that are supported by DB2 CLI (subset 1 of 2).
- SQL_CA1_NEXT
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2 (32-bit mask)
Indicates the attributes of a forward-only cursor that are supported by DB2 CLI (subset 2 of 2).
- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_OPT_ROWVER_CONCURRENCY
- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_GETDATA_EXTENSIONS (32-bit mask)
Indicates whether extensions to the `SQLGetData()` function are supported. The following extensions are currently identified and supported by DB2 CLI:
- SQL_GD_ANY_COLUMN, `SQLGetData()` can be called for unbound columns that precede the last bound column.
- SQL_GD_ANY_ORDER, `SQLGetData()` can be called for columns in any order.

ODBC also defines the following extensions which are not returned by DB2 CLI:
- SQL_GD_BLOCK
- SQL_GD_BOUND

SQL_GROUP_BY (16-bit integer)
Indicates the degree of support for the GROUP BY clause by the server:
- SQL_GB_NO_RELATION, there is no relationship between the columns in the GROUP BY and in the SELECT list
- SQL_GB_NOT_SUPPORTED, GROUP BY not supported
- SQL_GB_GROUP_BY_EQUALS_SELECT, GROUP BY must include all non-aggregated columns in the select list.
- SQL_GB_GROUP_BY_CONTAINS_SELECT, the GROUP BY clause must contain all non-aggregated columns in the SELECT list.
- SQL_GB_COLLATE, a COLLATE clause can be specified at the end of each grouping column.

SQL_IDENTIFIER_CASE (16-bit integer)
Indicates case sensitivity of object names (such as table-name).

A value of:
- SQL_IC_UPPER = identifier names are stored in uppercase in the system catalog.
- SQL_IC_LOWER = identifier names are stored in lowercase in the system catalog.
- SQL_IC_SENSITIVE = identifier names are case sensitive, and are stored in mixed case in the system catalog.
- SQL_IC_MIXED = identifier names are not case sensitive, and are stored in mixed case in the system catalog.

Note: Identifier names in IBM DBMSs are not case sensitive.

SQL_IDENTIFIER_QUOTE_CHAR (string)
Indicates the character used to surround a delimited identifier

SQL_INDEX_KEYWORDS (32-bit mask)
Indicates the keywords in the CREATE INDEX statement that are supported:
- SQL_IK_NONE, none of the keywords are supported.
- SQL_IK_ASC, ASC keyword is supported.
- SQL_IK_DESC, DESC keyword is supported.
- SQL_IK_ALL, all keywords are supported.

To see if the CREATE INDEX statement is supported, an application can call `SQLGetInfo()` with the SQL_DLL_INDEX *InfoType*.

SQL_INFO_SCHEMA_VIEWS (32-bit mask)
Indicates the views in the INFORMATION_SCHEMA that are supported. DB2 CLI always returns zero; no views in the INFORMATION_SCHEMA are supported.

ODBC also defines the following values that are not returned by DB2 CLI:
- SQL_ISV_ASSERTIONS
- SQL_ISV_CHARACTER_SETS
- SQL_ISV_CHECK_CONSTRAINTS
- SQL_ISV_COLLATIONS
- SQL_ISV_COLUMN_DOMAIN_USAGE
- SQL_ISV_COLUMN_PRIVILEGES
- SQL_ISV_COLUMNS
- SQL_ISV_CONSTRAINT_COLUMN_USAGE

- SQL_ISV_CONSTRAINT_TABLE_USAGE
- SQL_ISV_DOMAIN_CONSTRAINTS
- SQL_ISV_DOMAINS
- SQL_ISV_KEY_COLUMN_USAGE
- SQL_ISV_REFERENTIAL_CONSTRAINTS
- SQL_ISV_SCHEMATA
- SQL_ISV_SQL_LANGUAGES
- SQL_ISV_TABLE_CONSTRAINTS
- SQL_ISV_TABLE_PRIVILEGES
- SQL_ISV_TABLES
- SQL_ISV_TRANSLATIONS
- SQL_ISV_USAGE_PRIVILEGES
- SQL_ISV_VIEW_COLUMN_USAGE
- SQL_ISV_VIEW_TABLE_USAGE
- SQL_ISV_VIEWS

SQL_INSERT_STATEMENT (32-bit mask)
Indicates support for INSERT statements:
- SQL_IS_INSERT_LITERALS
- SQL_IS_INSERT_SEARCHED
- SQL_IS_SELECT_INTO

SQL_INTEGRITY (string)
The ″Y″ character string indicates that the data source supports Integrity Enhanced Facility (IEF) in SQL89 and in X/Open XPG4 Embedded SQL, an ″N″ indicates it does not.

In previous versions of DB2 CLI this *InfoType* was SQL_ODBC_SQL_OPT_IEF.

SQL_KEYSET_CURSOR_ATTRIBUTES1 (32-bit mask)
Indicates the attributes of a keyset cursor that are supported by DB2 CLI (subset 1 of 2).
- SQL_CA1_NEXT
- SQL_CA1_ABSOLUTE
- SQL_CA1_RELATIVE
- SQL_CA1_BOOKMARK
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_KEYSET_CURSOR_ATTRIBUTES2 (32-bit mask)
Indicates the attributes of a keyset cursor that are supported by DB2 CLI (subset 2 of 2).
- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_OPT_ROWVER_CONCURRENCY

- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_KEYWORDS (string)
A character string containing a comma-separated list of all data source-specific keywords. This is a list of all reserved keywords. Interoperable applications should not use these keywords in object names. This list does not contain keywords specific to ODBC or keywords used by both the data source and ODBC.

SQL_LIKE_ESCAPE_CLAUSE (string)
A character string ″Y″ if the data source supports an escape character for the percent character (%) and underscore (_) character in a LIKE predicate, and the driver supports the ODBC syntax for defining a LIKE predicate escape character; ″N″ otherwise.

SQL_LOCK_TYPES (32-bit mask)
Reserved option, zero is returned for the bit-mask.

SQL_MAX_ASYNC_CONCURRENT_STATEMENTS (32-bit unsigned integer)
The maximum number of active concurrent statements in asynchronous mode that DB2 CLI can support on a given connection. This value is zero if there is no specific limit, or the limit is unknown.

SQL_MAX_BINARY_LITERAL_LEN (32-bit unsigned integer)
A 32-bit unsigned integer value specifying the maximum length (number of hexadecimal characters, excluding the literal prefix and suffix returned by `SQLGetTypeInfo()`) of a binary literal in an SQL statement. For example, the binary literal 0xFFAA has a length of 4. If there is no maximum length or the length is unknown, this value is set to zero.

SQL_MAX_CATALOG_NAME_LEN (16-bit integer)
The maximum length of a catalog name in the data source. This value is zero if there is no maximum length, or the length is unknown.

In previous versions of DB2 CLI this *fInfoType* was SQL_MAX_QUALIFIER_NAME_LEN.

SQL_MAX_CHAR_LITERAL_LEN (32-bit unsigned integer)
The maximum length of a character literal in an SQL statement (in bytes). Zero if there is no limit.

SQL_MAX_COLUMN_NAME_LEN (16-bit integer)
The maximum length of a column name (in bytes). Zero if there is no limit.

SQL_MAX_COLUMNS_IN_GROUP_BY (16-bit integer)
Indicates the maximum number of columns that the server supports in a GROUP BY clause. Zero if no limit.

SQL_MAX_COLUMNS_IN_INDEX (16-bit integer)
Indicates the maximum number of columns that the server supports in an index. Zero if no limit.

SQL_MAX_COLUMNS_IN_ORDER_BY (16-bit integer)
Indicates the maximum number of columns that the server supports in an ORDER BY clause. Zero if no limit.

SQL_MAX_COLUMNS_IN_SELECT (16-bit integer)
Indicates the maximum number of columns that the server supports in a select list. Zero if no limit.

SQL_MAX_COLUMNS_IN_TABLE (16-bit integer)
Indicates the maximum number of columns that the server supports in a base table. Zero if no limit.

SQL_MAX_CONCURRENT_ACTIVITIES (16-bit integer)
The maximum number of active environments that the DB2 CLI driver can support. If there is no specified limit or the limit is unknown, this value is set to zero.

In previous versions of DB2 CLI this *InfoType* was SQL_ACTIVE_ENVIRONMENTS.

SQL_MAX_CURSOR_NAME_LEN (16-bit integer)
The maximum length of a cursor name (in bytes). This value is zero if there is no maximum length, or the length is unknown.

SQL_MAX_DRIVER_CONNECTIONS (16-bit integer)
The maximum number of active connections supported per application.

Zero is returned, indicating that the limit is dependent on system resources.

In previous versions of DB2 CLI this *InfoType* was SQL_ACTIVE_CONNECTIONS.

SQL_MAX_IDENTIFIER_LEN (16-bit integer)
The maximum size (in characters) that the data source supports for user-defined names.

SQL_MAX_INDEX_SIZE (32-bit unsigned integer)
Indicates the maximum size in bytes that the server supports for the combined columns in an index. Zero if no limit.

SQL_MAX_PROCEDURE_NAME_LEN (16-bit integer)
The maximum length of a procedure name (in bytes).

SQL_MAX_ROW_SIZE (32-bit unsigned integer)
Specifies the maximum length in bytes that the server supports in single row of a base table. Zero if no limit.

SQL_MAX_ROW_SIZE_INCLUDES_LONG (string)
Set to "Y" to indicate that the value returned by SQL_MAX_ROW_SIZE *InfoType* includes the length of product-specific *long string* data types. Otherwise, set to "N".

SQL_MAX_SCHEMA_NAME_LEN (16-bit integer)
The maximum length of a schema qualifier name (in bytes).

In previous versions of DB2 CLI this *fInfoType* was SQL_MAX_OWNER_NAME_LEN.

SQL_MAX_STATEMENT_LEN (32-bit unsigned integer)
Indicates the maximum length of an SQL statement string in bytes, including the number of white spaces in the statement.

SQL_MAX_TABLE_NAME_LEN (16-bit integer)
The maximum length of a table name (in bytes).

SQL_MAX_TABLES_IN_SELECT (16-bit integer)
Indicates the maximum number of table names allowed in a FROM clause in a <query specification>.

SQL_MAX_USER_NAME_LEN (16-bit integer)
Indicates the maximum size allowed for a <user identifier> (in bytes).

SQL_MULT_RESULT_SETS (string)
The character string ″Y″ indicates that the database supports multiple result sets, ″N″ indicates that it does not.

SQL_MULTIPLE_ACTIVE_TXN (string)
The character string ″Y″ indicates that active transactions on multiple connections are allowed, ″N″ indicates that only one connection at a time can have an active transaction.

DB2 CLI returns ″N″ for coordinated distributed unit of work (CONNECT TYPE 2) connections, (since the transaction or Unit Of Work spans all connections), and returns ″Y″ for all other connections.

SQL_NEED_LONG_DATA_LEN (string)
A character string reserved for the use of ODBC. “N” is always returned.

SQL_NON_NULLABLE_COLUMNS (16-bit integer)
Indicates whether non-nullable columns are supported:
- SQL_NNC_NON_NULL, columns can be defined as NOT NULL.
- SQL_NNC_NULL, columns can not be defined as NOT NULL.

SQL_NULL_COLLATION (16-bit integer)
Indicates where NULLs are sorted in a result set:
- SQL_NC_HIGH, null values sort high
- SQL_NC_LOW, to indicate that null values sort low

SQL_NUMERIC_FUNCTIONS (32-bit mask)
Indicates the ODBC scalar numeric functions supported These functions are intended to be used with the ODBC vendor escape sequence.

The following bit-masks are used to determine which numeric functions are supported:
- SQL_FN_NUM_ABS
- SQL_FN_NUM_ACOS
- SQL_FN_NUM_ASIN
- SQL_FN_NUM_ATAN
- SQL_FN_NUM_ATAN2
- SQL_FN_NUM_CEILING
- SQL_FN_NUM_COS
- SQL_FN_NUM_COT
- SQL_FN_NUM_DEGREES
- SQL_FN_NUM_EXP
- SQL_FN_NUM_FLOOR
- SQL_FN_NUM_LOG
- SQL_FN_NUM_LOG10
- SQL_FN_NUM_MOD
- SQL_FN_NUM_PI

- SQL_FN_NUM_POWER
- SQL_FN_NUM_RADIANS
- SQL_FN_NUM_RAND
- SQL_FN_NUM_ROUND
- SQL_FN_NUM_SIGN
- SQL_FN_NUM_SIN
- SQL_FN_NUM_SQRT
- SQL_FN_NUM_TAN
- SQL_FN_NUM_TRUNCATE

SQL_ODBC_API_CONFORMANCE (16-bit integer)
The level of ODBC conformance.
- SQL_OAC_NONE
- SQL_OAC_LEVEL1
- SQL_OAC_LEVEL2

SQL_ODBC_INTERFACE_CONFORMANCE (32-bit unsigned integer)
Indicates the level of the ODBC 3.0 interface that the DB2 CLI driver conforms to:
- SQL_OIC_CORE, the minimum level that all ODBC drivers are expected to conform to. This level includes basic interface elements such as connection functions; functions for preparing and executing an SQL statement; basic result set metadata functions; basic catalog functions; and so on.
- SQL_OIC_LEVEL1, a level including the core standards compliance level functionality, plus scrollable cursors, bookmarks, positioned updates and deletes, and so on.
- SQL_OIC_LEVEL2, a level including level 1 standards compliance level functionality, plus advanced features such as sensitive cursors; update, delete, and refresh by bookmarks; stored procedure support; catalog functions for primary and foreign keys; multi-catalog support; and so on.

SQL_ODBC_SAG_CLI_CONFORMANCE (16-bit integer)
The compliance to the functions of the SQL Access Group (SAG) CLI specification.

A value of:
- SQL_OSCC_NOT_COMPLIANT - the driver is not SAG-compliant.
- SQL_OSCC_COMPLIANT - the driver is SAG-compliant.

SQL_ODBC_SQL_CONFORMANCE (16-bit integer)
A value of:
- SQL_OSC_MINIMUM, minimum ODBC SQL grammar supported
- SQL_OSC_CORE, core ODBC SQL Grammar supported
- SQL_OSC_EXTENDED, extended ODBC SQL Grammar supported

SQL_ODBC_VER (string)
The version number of ODBC that the driver manager supports.

DB2 CLI will return the string ″03.01.0000″.

SQL_OJ_CAPABILITIES (32-bit mask)
A 32-bit bit-mask enumerating the types of outer join supported.

The bitmasks are:
- SQL_OJ_LEFT : Left outer join is supported.
- SQL_OJ_RIGHT : Right outer join is supported.
- SQL_OJ_FULL : Full outer join is supported.
- SQL_OJ_NESTED : Nested outer join is supported.

- SQL_OJ_ORDERED : The order of the tables underlying the columns in the outer join ON clause need not be in the same order as the tables in the JOIN clause.
- SQL_OJ_INNER : The inner table of an outer join can also be an inner join.
- SQL_OJ_ALL_COMPARISONS_OPS : Any predicate can be used in the outer join ON clause. If this bit is not set, only the equality (=) comparison operator can be used in outer joins.

SQL_ORDER_BY_COLUMNS_IN_SELECT (string)
Set to ″Y″ if columns in the ORDER BY clauses must be in the select list; otherwise set to ″N″.

SQL_OUTER_JOINS (string)
The character string:
- ″Y″ indicates that outer joins are supported, and DB2 CLI supports the ODBC outer join request syntax.
- ″N″ indicates that it is not supported.

SQL_PARAM_ARRAY_ROW_COUNTS (32-bit unsigned integer)
Indicates the availability of row counts in a parameterized execution:
- SQL_PARC_BATCH, individual row counts are available for each set of parameters. This is conceptually equivalent to the driver generating a batch of SQL statements, one for each parameter set in the array. Extended error information can be retrieved by using the SQL_PARAM_STATUS_PTR descriptor field.
- SQL_PARC_NO_BATCH, there is only one row count available, which is the cumulative row count resulting from the execution of the statement for the entire array of parameters. This is conceptually equivalent to treating the statement along with the entire parameter array as one atomic unit. Errors are handled the same as if one statement were executed.

SQL_PARAM_ARRAY_SELECTS (32-bit unsigned integer)
Indicates the availability of result sets in a parameterized execution:
- SQL_PAS_BATCH, there is one result set available per set of parameters. This is conceptually equivalent to the driver generating a batch of SQL statements, one for each parameter set in the array.
- SQL_PAS_NO_BATCH, there is only one result set available, which represents the cumulative result set resulting from the execution of the statement for the entire array of parameters. This is conceptually equivalent to treating the statement along with the entire parameter array as one atomic unit.
- SQL_PAS_NO_SELECT, a driver does not allow a result-set generating statement to be executed with an array of parameters.

SQL_POS_OPERATIONS (32-bit mask)
Reserved option, zero is returned for the bit-mask.

SQL_POSITIONED_STATEMENTS (32-bit mask)
Indicates the degree of support for Positioned UPDATE and Positioned DELETE statements:
- SQL_PS_POSITIONED_DELETE
- SQL_PS_POSITIONED_UPDATE
- SQL_PS_SELECT_FOR_UPDATE, indicates whether or not the server requires the FOR UPDATE clause to be specified on a <query expression> in order for a column to be updateable via the cursor.

SQL_PROCEDURE_TERM (string)
The name a database vendor uses for a procedure

SQL_PROCEDURES (string)
A character string of "Y" indicates that the data source supports procedures and DB2 CLI supports the ODBC procedure invocation syntax specified by the CALL statement. "N" indicates that it does not.

SQL_QUOTED_IDENTIFIER_CASE (16-bit integer)
Returns:
- SQL_IC_UPPER - quoted identifiers in SQL are case insensitive and stored in uppercase in the system catalog.
- SQL_IC_LOWER - quoted identifiers in SQL are case insensitive and are stored in lowercase in the system catalog.
- SQL_IC_SENSITIVE - quoted identifiers (delimited identifiers) in SQL are case sensitive and are stored in mixed case in the system catalog.
- SQL_IC_MIXED - quoted identifiers in SQL are case insensitive and are stored in mixed case in the system catalog.

This should be contrasted with the SQL_IDENTIFIER_CASE *InfoType* which is used to determine how (unquoted) identifiers are stored in the system catalog.

SQL_ROW_UPDATES (string)
A character string of "Y" indicates a keyset-driven or mixed cursor maintains row versions or values for all fetched rows and therefore can only detect any updates made to a row by any user since the row was last fetched. (This only applies to updates, not to deletions or insertions.) The driver can return the SQL_ROW_UPDATED flag to the row status array when `SQLFetchScroll()` is called. Otherwise, "N".

SQL_SCHEMA_TERM (string)
The database vendor's terminology for a schema (owner).

In previous versions of DB2 CLI this *InfoType* was SQL_OWNER_TERM.

SQL_SCHEMA_USAGE (32-bit mask)
Indicates the type of SQL statements that have schema (owners) associated with them when these statements are executed, Schema qualifiers (owners) are:
- SQL_SU_DML_STATEMENTS - supported in all DML statements.
- SQL_SU_PROCEDURE_INVOCATION - supported in the procedure invocation statement.
- SQL_SU_TABLE_DEFINITION - supported in all table definition statements.
- SQL_SU_INDEX_DEFINITION - supported in all index definition statements.
- SQL_SU_PRIVILEGE_DEFINITION - supported in all privilege definition statements (that is, in grant and revoke statements).

In previous versions of DB2 CLI this *InfoType* was SQL_OWNER_USAGE.

SQL_SCROLL_CONCURRENCY (32-bit mask)
Indicates the concurrency options supported for the cursor.

The following bit-masks are used in conjunction with the flag to determine which options are supported:
- SQL_SCCO_READ_ONLY
- SQL_SCCO_LOCK
- SQL_SCCO_TIMESTAMP
- SQL_SCCO_VALUES

DB2 CLI returns SQL_SCCO_LOCK. indicating that the lowest level of locking that is sufficient to ensure the row can be updated is used.

SQL_SCROLL_OPTIONS (32-bit mask)
The scroll options supported for scrollable cursors.

The following bit-masks are used in conjunction with the flag to determine which options are supported:
- SQL_SO_FORWARD_ONLY: The cursor only scrolls forward.
- SQL_SO_KEYSET_DRIVEN: The driver saves and uses the keys for every row in the result set.
- SQL_SO_STATIC: The data in the result set is static.
- SQL_SO_DYNAMIC: The driver keeps the keys for every row in the rowset (the keyset size is the same as the rowset size).
- SQL_SO_MIXED: The driver keeps the keys for every row in the keyset, and the keyset size is greater than the rowset size. The cursor is keyset-driven inside the keyset and dynamic outside the keyset.

SQL_SEARCH_PATTERN_ESCAPE (string)
Used to specify what the driver supports as an escape character for catalog functions such as `SQLTables()`, `SQLColumns()`.

SQL_SERVER_NAME (string)
The Name of the DB2 Instance. In contrast to SQL_DATA_SOURCE_NAME, this is the actual name of the database server. (Some DBMSs provide a different name on CONNECT than the real server-name of the database.)

SQL_SPECIAL_CHARACTERS (string)
A character string containing all special characters (that is, all characters except a...z, A...Z, 0...9, and underscore) that can be used in an identifier name, such as table, column, or index name, on the data source. For example, "@#". If an identifier contains one or more of these characters, the identifier must be a delimited identifier.

SQL_SQL_CONFORMANCE (32-bit unsigned integer)
Indicates the level of SQL-92 supported:
- SQL_SC_SQL92_ENTRY, entry level SQL-92 compliant.
- SQL_SC_FIPS127_2_TRANSITIONAL, FIPS 127-2 transitional level compliant.
- SQL_SC_SQL92_FULL, full level SQL-92 compliant.
- SQL_SC_ SQL92_INTERMEDIATE, intermediate level SQL-92 compliant.

SQL_SQL92_DATETIME_FUNCTIONS (32-bit mask)
Indicates the datetime scalar functions that are supported by DB2 CLI and the data source:
- SQL_SDF_CURRENT_DATE
- SQL_SDF_CURRENT_TIME
- SQL_SDF_CURRENT_TIMESTAMP

SQL_SQL92_FOREIGN_KEY_DELETE_RULE (32-bit mask)
Indicates the rules supported for a foreign key in a DELETE statement, as defined by SQL-92:
- SQL_SFKD_CASCADE
- SQL_SFKD_NO_ACTION
- SQL_SFKD_SET_DEFAULT
- SQL_SFKD_SET_NULL

SQL_SQL92_FOREIGN_KEY_UPDATE_RULE (32-bit mask)
Indicates the rules supported for a foreign key in an UPDATE statement, as defined by SQL-92:
- SQL_SFKU_CASCADE
- SQL_SFKU_NO_ACTION
- SQL_SFKU_SET_DEFAULT
- SQL_SFKU_SET_NULL

SQL_SQL92_GRANT (32-bit mask)
Indicates the clauses supported in a GRANT statement, as defined by SQL-92:
- SQL_SG_DELETE_TABLE
- SQL_SG_INSERT_COLUMN
- SQL_SG_INSERT_TABLE
- SQL_SG_REFERENCES_TABLE
- SQL_SG_REFERENCES_COLUMN
- SQL_SG_SELECT_TABLE
- SQL_SG_UPDATE_COLUMN
- SQL_SG_UPDATE_TABLE
- SQL_SG_USAGE_ON_DOMAIN
- SQL_SG_USAGE_ON_CHARACTER_SET
- SQL_SG_USAGE_ON_COLLATION
- SQL_SG_USAGE_ON_TRANSLATION
- SQL_SG_WITH_GRANT_OPTION

SQL_SQL92_NUMERIC_VALUE_FUNCTIONS (32-bit mask)
Indicates the numeric value scalar functions that are supported by DB2 CLI and the data source, as defined in SQL-92:
- SQL_SNVF_BIT_LENGTH
- SQL_SNVF_CHAR_LENGTH
- SQL_SNVF_CHARACTER_LENGTH
- SQL_SNVF_EXTRACT
- SQL_SNVF_OCTET_LENGTH
- SQL_SNVF_POSITION

SQL_SQL92_PREDICATES (32-bit mask)
Indicates the predicates supported in a SELECT statement, as defined by SQL-92.
- SQL_SP_BETWEEN
- SQL_SP_COMPARISON
- SQL_SP_EXISTS
- SQL_SP_IN
- SQL_SP_ISNOTNULL
- SQL_SP_ISNULL
- SQL_SP_LIKE
- SQL_SP_MATCH_FULL
- SQL_SP_MATCH_PARTIAL
- SQL_SP_MATCH_UNIQUE_FULL
- SQL_SP_MATCH_UNIQUE_PARTIAL
- SQL_SP_OVERLAPS
- SQL_SP_QUANTIFIED_COMPARISON
- SQL_SP_UNIQUE

SQL_SQL92_RELATIONAL_JOIN_OPERATORS (32-bit mask)
Indicates the relational join operators supported in a SELECT statement, as defined by SQL-92.
- SQL_SRJO_CORRESPONDING_CLAUSE
- SQL_SRJO_CROSS_JOIN

- SQL_SRJO_EXCEPT_JOIN
- SQL_SRJO_FULL_OUTER_JOIN
- SQL_SRJO_INNER_JOIN (indicates support for the INNER JOIN syntax, not for the inner join capability)
- SQL_SRJO_INTERSECT_JOIN
- SQL_SRJO_LEFT_OUTER_JOIN
- SQL_SRJO_NATURAL_JOIN
- SQL_SRJO_RIGHT_OUTER_JOIN
- SQL_SRJO_UNION_JOIN

SQL_SQL92_REVOKE (32-bit mask)
Indicates which clauses the data source supports in the REVOKE statement, as defined by SQL-92:
- SQL_SR_CASCADE
- SQL_SR_DELETE_TABLE
- SQL_SR_GRANT_OPTION_FOR
- SQL_SR_INSERT_COLUMN
- SQL_SR_INSERT_TABLE
- SQL_SR_REFERENCES_COLUMN
- SQL_SR_REFERENCES_TABLE
- SQL_SR_RESTRICT
- SQL_SR_SELECT_TABLE
- SQL_SR_UPDATE_COLUMN
- SQL_SR_UPDATE_TABLE
- SQL_SR_USAGE_ON_DOMAIN
- SQL_SR_USAGE_ON_CHARACTER_SET
- SQL_SR_USAGE_ON_COLLATION
- SQL_SR_USAGE_ON_TRANSLATION

SQL_SQL92_ROW_VALUE_CONSTRUCTOR (32-bit mask)
Indicates the row value constructor expressions supported in a SELECT statement, as defined by SQL-92.
- SQL_SRVC_VALUE_EXPRESSION
- SQL_SRVC_NULL
- SQL_SRVC_DEFAULT
- SQL_SRVC_ROW_SUBQUERY

SQL_SQL92_STRING_FUNCTIONS (32-bit mask)
Indicates the string scalar functions that are supported by DB2 CLI and the data source, as defined by SQL-92:
- SQL_SSF_CONVERT
- SQL_SSF_LOWER
- SQL_SSF_UPPER
- SQL_SSF_SUBSTRING
- SQL_SSF_TRANSLATE
- SQL_SSF_TRIM_BOTH
- SQL_SSF_TRIM_LEADING
- SQL_SSF_TRIM_TRAILING

SQL_SQL92_VALUE_EXPRESSIONS (32-bit mask)
Indicates the value expressions supported, as defined by SQL-92.
- SQL_SVE_CASE
- SQL_SVE_CAST
- SQL_SVE_COALESCE
- SQL_SVE_NULLIF

SQL_STANDARD_CLI_CONFORMANCE (32-bit mask)
Indicates the CLI standard or standards to which DB2 CLI conforms:

- SQL_SCC_XOPEN_CLI_VERSION1
- SQL_SCC_ISO92_CLI

SQL_STATIC_CURSOR_ATTRIBUTES1 (32-bit mask)
Indicates the attributes of a static cursor that are supported by DB2 CLI (subset 1 of 2):
- SQL_CA1_NEXT
- SQL_CA1_ABSOLUTE
- SQL_CA1_RELATIVE
- SQL_CA1_BOOKMARK
- SQL_CA1_LOCK_NO_CHANGE
- SQL_CA1_LOCK_EXCLUSIVE
- SQL_CA1_LOCK_UNLOCK
- SQL_CA1_POS_POSITION
- SQL_CA1_POS_UPDATE
- SQL_CA1_POS_DELETE
- SQL_CA1_POS_REFRESH
- SQL_CA1_POSITIONED_UPDATE
- SQL_CA1_POSITIONED_DELETE
- SQL_CA1_SELECT_FOR_UPDATE
- SQL_CA1_BULK_ADD
- SQL_CA1_BULK_UPDATE_BY_BOOKMARK
- SQL_CA1_BULK_DELETE_BY_BOOKMARK
- SQL_CA1_BULK_FETCH_BY_BOOKMARK

SQL_STATIC_CURSOR_ATTRIBUTES2 (32-bit mask)
Indicates the attributes of a static cursor that are supported by DB2 CLI (subset 2 of 2):
- SQL_CA2_READ_ONLY_CONCURRENCY
- SQL_CA2_LOCK_CONCURRENCY
- SQL_CA2_OPT_ROWVER_CONCURRENCY
- SQL_CA2_OPT_VALUES_CONCURRENCY
- SQL_CA2_SENSITIVITY_ADDITIONS
- SQL_CA2_SENSITIVITY_DELETIONS
- SQL_CA2_SENSITIVITY_UPDATES
- SQL_CA2_MAX_ROWS_SELECT
- SQL_CA2_MAX_ROWS_INSERT
- SQL_CA2_MAX_ROWS_DELETE
- SQL_CA2_MAX_ROWS_UPDATE
- SQL_CA2_MAX_ROWS_CATALOG
- SQL_CA2_MAX_ROWS_AFFECTS_ALL
- SQL_CA2_CRC_EXACT
- SQL_CA2_CRC_APPROXIMATE
- SQL_CA2_SIMULATE_NON_UNIQUE
- SQL_CA2_SIMULATE_TRY_UNIQUE
- SQL_CA2_SIMULATE_UNIQUE

SQL_STATIC_SENSITIVITY (32-bit mask)
Indicates whether changes made by an application with a positioned update or delete statement can be detected by that application:
- SQL_SS_ADDITIONS: Added rows are visible to the cursor; the cursor can scroll to these rows. All DB2 servers see added rows.
- SQL_SS_DELETIONS: Deleted rows are no longer available to the cursor and do not leave a hole in the result set; after the cursor scrolls from a deleted row, it cannot return to that row.

- SQL_SS_UPDATES: Updates to rows are visible to the cursor; if the cursor scrolls from and returns to an updated row, the data returned by the cursor is the updated data, not the original data.

SQL_STRING_FUNCTIONS (32-bit mask)
Indicates which string functions are supported.

The following bit-masks are used to determine which string functions are supported:
- SQL_FN_STR_ASCII
- SQL_FN_STR_BIT_LENGTH
- SQL_FN_STR_CHAR
- SQL_FN_STR_CHAR_LENGTH
- SQL_FN_STR_CHARACTER_LENGTH
- SQL_FN_STR_CONCAT
- SQL_FN_STR_DIFFERENCE
- SQL_FN_STR_INSERT
- SQL_FN_STR_LCASE
- SQL_FN_STR_LEFT
- SQL_FN_STR_LENGTH
- SQL_FN_STR_LOCATE
- SQL_FN_STR_LOCATE_2
- SQL_FN_STR_LTRIM
- SQL_FN_STR_OCTET_LENGTH
- SQL_FN_STR_POSITION
- SQL_FN_STR_REPEAT
- SQL_FN_STR_REPLACE
- SQL_FN_STR_RIGHT
- SQL_FN_STR_RTRIM
- SQL_FN_STR_SOUNDEX
- SQL_FN_STR_SPACE
- SQL_FN_STR_SUBSTRING
- SQL_FN_STR_UCASE

If an application can call the LOCATE scalar function with the *string_exp1*, *string_exp2*, and *start* arguments, the SQL_FN_STR_LOCATE bitmask is returned. If an application can only call the LOCATE scalar function with the *string_exp1* and *string_exp2*, the SQL_FN_STR_LOCATE_2 bitmask is returned. If the LOCATE scalar function is fully supported, both bitmasks are returned.

SQL_SUBQUERIES (32-bit mask)
Indicates which predicates support subqueries:
- SQL_SQ_COMPARISION - the *comparison* predicate
- SQL_SQ_CORRELATE_SUBQUERIES - all predicates that support subqueries support correlated subqueries
- SQL_SQ_EXISTS - the *exists* predicate
- SQL_SQ_IN - the *in* predicate
- SQL_SQ_QUANTIFIED - the predicates containing a quantification scalar function.

SQL_SYSTEM_FUNCTIONS (32-bit mask)
Indicates which scalar system functions are supported.

The following bit-masks are used to determine which scalar system functions are supported:
- SQL_FN_SYS_DBNAME
- SQL_FN_SYS_IFNULL

- SQL_FN_SYS_USERNAME

Note: These functions are intended to be used with the escape sequence in ODBC.

SQL_TABLE_TERM (string)
The database vendor's terminology for a table

SQL_TIMEDATE_ADD_INTERVALS (32-bit mask)
Indicates whether or not the special ODBC system function TIMESTAMPADD is supported, and, if it is, which intervals are supported.

The following bitmasks are used to determine which intervals are supported:
- SQL_FN_TSI_FRAC_SECOND
- SQL_FN_TSI_SECOND
- SQL_FN_TSI_MINUTE
- SQL_FN_TSI_HOUR
- SQL_FN_TSI_DAY
- SQL_FN_TSI_WEEK
- SQL_FN_TSI_MONTH
- SQL_FN_TSI_QUARTER
- SQL_FN_TSI_YEAR

SQL_TIMEDATE_DIFF_INTERVALS (32-bit mask)
Indicates whether or not the special ODBC system function TIMESTAMPDIFF is supported, and, if it is, which intervals are supported.

The following bitmasks are used to determine which intervals are supported:
- SQL_FN_TSI_FRAC_SECOND
- SQL_FN_TSI_SECOND
- SQL_FN_TSI_MINUTE
- SQL_FN_TSI_HOUR
- SQL_FN_TSI_DAY
- SQL_FN_TSI_WEEK
- SQL_FN_TSI_MONTH
- SQL_FN_TSI_QUARTER
- SQL_FN_TSI_YEAR

SQL_TIMEDATE_FUNCTIONS (32-bit mask)
Indicates which time and date functions are supported.

The following bit-masks are used to determine which date functions are supported:
- SQL_FN_TD_CURRENT_DATE
- SQL_FN_TD_CURRENT_TIME
- SQL_FN_TD_CURRENT_TIMESTAMP
- SQL_FN_TD_CURDATE
- SQL_FN_TD_CURTIME
- SQL_FN_TD_DAYNAME
- SQL_FN_TD_DAYOFMONTH
- SQL_FN_TD_DAYOFWEEK
- SQL_FN_TD_DAYOFYEAR
- SQL_FN_TD_EXTRACT
- SQL_FN_TD_HOUR
- SQL_FN_TD_JULIAN_DAY
- SQL_FN_TD_MINUTE
- SQL_FN_TD_MONTH

- SQL_FN_TD_MONTHNAME
- SQL_FN_TD_NOW
- SQL_FN_TD_QUARTER
- SQL_FN_TD_SECOND
- SQL_FN_TD_SECONDS_SINCE_MIDNIGHT
- SQL_FN_TD_TIMESTAMPADD
- SQL_FN_TD_TIMESTAMPDIFF
- SQL_FN_TD_WEEK
- SQL_FN_TD_YEAR

Note: These functions are intended to be used with the escape sequence in ODBC.

SQL_TXN_CAPABLE (16-bit integer)
Indicates whether transactions can contain DDL or DML or both.
- SQL_TC_NONE = transactions not supported.
- SQL_TC_DML = transactions can only contain DML statements (for example, SELECT, INSERT, UPDATE and DELETE). DDL statements (for example, CREATE TABLE and DROP INDEX) encountered in a transaction cause an error.
- SQL_TC_DDL_COMMIT = transactions can only contain DML statements. DDL statements encountered in a transaction cause the transaction to be committed.
- SQL_TC_DDL_IGNORE = transactions can only contain DML statements. DDL statements encountered in a transaction are ignored.
- SQL_TC_ALL = transactions can contain DDL and DML statements in any order.

SQL_TXN_ISOLATION_OPTION (32-bit mask)
The transaction isolation levels available at the currently connected database server.

The following masks are used in conjunction with the flag to determine which options are supported:
- SQL_TXN_READ_UNCOMMITTED
- SQL_TXN_READ_COMMITTED
- SQL_TXN_REPEATABLE_READ
- SQL_TXN_SERIALIZABLE
- SQL_TXN_NOCOMMIT
- SQL_TXN_VERSIONING

For descriptions of each level refer to SQL_DEFAULT_TXN_ISOLATION.

SQL_UNION (32-bit mask)
Indicates if the server supports the UNION operator:
- SQL_U_UNION - supports the UNION clause
- SQL_U_UNION_ALL - supports the ALL keyword in the UNION clause

If SQL_U_UNION_ALL is set, so is SQL_U_UNION.

SQL_USER_NAME (string)
The user name used in a particular database. This is the identifier specified on the `SQLConnect()` call.

SQL_XOPEN_CLI_YEAR (string)
Indicates the year of publication of the X/Open specification with which the version of the driver fully complies.

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Vendor escape clauses in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CALL statement" in *SQL Reference, Volume 2*
- "SQLGetTypeInfo function (CLI) - Get data type information" on page 224
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "ilinfo.c -- How to get information at the installation image level"
- "ininfo.c -- How to get information at the instance level"

SQLGetLength function (CLI) - Retrieve length of a string value

Purpose:

Specification:	**DB2 CLI** 2.1		

`SQLGetLength()` is used to retrieve the length of a large object value, referenced by a large object locator that has been returned from the server (as a result of a fetch, or an `SQLGetSubString()` call) during the current transaction.

Syntax:

```
SQLRETURN  SQLGetLength   (SQLHSTMT          StatementHandle,  /* hstmt */
                           SQLSMALLINT       LocatorCType,
                           SQLINTEGER        Locator,
                           SQLINTEGER        *StringLength,
                           SQLINTEGER        *IndicatorValue);
```

Function arguments:

Table 91. SQLGetLength arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	*LocatorCType*	input	The C type of the source LOB locator. This may be: • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	input	Must be set to the LOB locator value.

Table 91. SQLGetLength arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	*StringLength*	output	The length of the returned information in *rgbValue* in bytes[a] if the target C buffer type is intended for a binary or character string variable and not a locator value. If the pointer is set to NULL then the SQLSTATE **HY**009 is returned.
SQLINTEGER *	*IndicatorValue*	output	Always set to zero.

Note:

a This is in characters for DBCLOB data.

Usage:

`SQLGetLength()` can be used to determine the length of the data value represented by a LOB locator. It is used by applications to determine the overall length of the referenced LOB value so that the appropriate strategy to obtain some or all of the LOB value can be chosen. The length is calculated by the database server using the server code page, and so if the application code page is different from the server code page, then there may be some complexity in calculating space requirements on the client. The application will need to allow for code page expansion if any is needed.

The *Locator* argument can contain any valid LOB locator which has not been explicitly freed using a FREE LOCATOR statement nor implicitly freed because the transaction during which it was created has ended.

The statement handle must not have been associated with any prepared statements or catalog function calls.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 92. SQLGetLength SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The combination of *LocatorCType* and *Locator* is not valid.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	*LocatorCType* is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.

Table 92. SQLGetLength SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY009	Invalid argument value.	Pointer to *StringLength* was NULL.
HY010	Function sequence error.	The specified *StatementHandle* is not in an *allocated* state. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	The LOB token variable does not currently represent any value.	The value specified for *Locator* has not been associated with a LOB locator.

Restrictions:

This function is not available when connected to a DB2 server that does not support large objects. Call `SQLGetFunctions()` with the function type set to SQL_API_SQLGETLENGTH and check the *fExists* output argument to determine if the function is supported for the current connection.

Example:

```
/* get the length of the whole CLOB data */
cliRC = SQLGetLength(hstmtLocUse,
                     SQL_C_CLOB_LOCATOR,
                     clobLoc,
                     &clobLen,
                     &ind);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Large object usage in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "LOB locators in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLGetSubString function (CLI) - Retrieve portion of a string value" on page 220

Related samples:

- "spserver.c -- Definition of various types of stored procedures"
- "dtlob.c -- How to read and write LOB data"

SQLGetPosition function (CLI) - Return starting position of string

Purpose:

Specification:	DB2 CLI 2.1		

`SQLGetPosition()` is used to return the starting position of one string within a LOB value (the source). The source value must be a LOB locator, the search string can be a LOB locator or a literal string.

The source and search LOB locators can be any that have been returned from the database from a fetch or a `SQLGetSubString()` call during the current transaction.

Syntax:

```
SQLRETURN   SQLGetPosition   (SQLHSTMT         StatementHandle,   /* hstmt */
                              SQLSMALLINT      LocatorCType,
                              SQLINTEGER       SourceLocator,
                              SQLINTEGER       SearchLocator,
                              SQLCHAR          *SearchLiteral,
                              SQLINTEGER       SearchLiteralLength,
                              SQLUINTEGER      FromPosition,
                              SQLUINTEGER      *LocatedAt,
                              SQLINTEGER       *IndicatorValue);
```

Function arguments:

Table 93. SQLGetPosition arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	*LocatorCType*	input	The C type of the source LOB locator. This can be: • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	input	*Locator* must be set to the source LOB locator.
SQLINTEGER	*SearchLocator*	input	If the *SearchLiteral* pointer is NULL and if *SearchLiteralLength* is set to 0, then *SearchLocator* must be set to the LOB locator associated with the search string; otherwise, this argument is ignored.
SQLCHAR *	*SearchLiteral*	input	This argument points to the area of storage that contains the search string literal. If *SearchLiteralLength* is 0, this pointer must be NULL.
SQLINTEGER	*SearchLiteralLength*	input	The length of the string in *SearchLiteral*(in bytes). [a] If this argument value is 0, then the argument *SearchLocator* is meaningful.
SQLUINTEGER	*FromPosition*	input	For BLOBs and CLOBs, this is the position of the first byte within the source string at which the search is to start. For DBCLOBs, this is the first character. The start byte or character is numbered 1.

Table 93. SQLGetPosition arguments (continued)

Data type	Argument	Use	Description
SQLUINTEGER *	*LocatedAt*	output	For BLOBs and CLOBs, this is the byte position at which the string was located or, if not located, the value zero. For DBCLOBs, this is the character position. If the length of the source string is zero, the value 1 is returned.
SQLINTEGER *	*IndicatorValue*	output	Always set to zero.

Note:

a This is in bytes even for DBCLOB data.

Usage:

`SQLGetPosition()` is used in conjunction with `SQLGetSubString()` in order to obtain any portion of a LOB in a random manner. In order to use `SQLGetSubString()`, the location of the substring within the overall string must be known in advance. In situations where the start of that substring can be found by a search string, `SQLGetPosition()` can be used to obtain the starting position of that substring.

The *Locator* and *SearchLocator* (if used) arguments can contain any valid LOB locator which has not been explicitly freed using a FREE LOCATOR statement or implicitly freed because the transaction during which it was created has ended.

The *Locator* and *SearchLocator* must have the same LOB locator type.

The statement handle must not have been associated with any prepared statements or catalog function calls.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 94. SQLGetPosition SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	The combination of *LocatorCType* and either of the LOB locator values is not valid.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

Table 94. SQLGetPosition SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY009	Invalid argument value.	The pointer to the *LocatedAt* argument was NULL. The argument value for *FromPosition* was not greater than 0. *LocatorCType* is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY010	Function sequence error.	The specified *StatementHandle* is not in an *allocated* state. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value of *SearchLiteralLength* was less than 1, and not SQL_NTS. The length of the pattern is longer than the maximum data length of the associated variable SQL data type (for DB2 UDB for z/OS and OS/390 servers, the pattern length is a maximum of 4000 bytes regardless of the data type or the *LocatorCType*). For LocatorCType of SQL_C_CLOB_LOCATOR, the literal maximum size is that of an SQLCLOB; for LocatorCType of SQL_C_BLOB_LOCATOR, the literal maximum size is that of an SQLVARBINARY; for LocatorCType of SQL_C_DBLOB_LOCATOR, the literal maximum size is that of an SQLVARGRAPHIC.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	The LOB token variable does not currently represent any value.	The value specified for *Locator* or *SearchLocator* is not currently a LOB locator.

Restrictions:

This function is not available when connected to a DB2 server that does not support large objects. Call `SQLGetFunctions()` with the function type set to SQL_API_SQLGETPOSITION and check the *fExists* output argument to determine if the function is supported for the current connection.

Example:

```
/* get the starting position of the CLOB piece of data */
cliRC = SQLGetPosition(hstmtLocUse,
                       SQL_C_CLOB_LOCATOR,
                       clobLoc,
                       0,
                       (SQLCHAR *)"Interests",
                       strlen("Interests"),
                       1,
                       &clobPiecePos,
                       &ind);
```

Related concepts:

- "Large object usage in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "LOB locators in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "FREE LOCATOR statement" in *SQL Reference, Volume 2*
- "SQLGetLength function (CLI) - Retrieve length of a string value" on page 210
- "SQLGetSubString function (CLI) - Retrieve portion of a string value" on page 220

Related samples:

- "spserver.c -- Definition of various types of stored procedures"
- "dtlob.c -- How to read and write LOB data"

SQLGetSQLCA function (CLI) - Get SQLCA data structure

Deprecated:

Note:

`SQLGetSQLCA()` has been deprecated.

Although this version of DB2 CLI continues to support `SQLGetSQLCA()`, it is recommended that you stop using it in your DB2 CLI programs so that they conform to the latest standards.

Use `SQLGetDiagField()` and `SQLGetDiagRec()` to retrieve diagnostic information.

Related concepts:

- "Diagnostics in CLI applications overview" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*
- "SQLGetDiagField function (CLI) - Get a field of diagnostic data" on page 168
- "SQLGetDiagRec function (CLI) - Get multiple fields settings of diagnostic record" on page 173

Related samples:

- "clisqlca.c -- How to retrieve SQLCA-equivalent information "

SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

SQLGetStmtAttr() returns the current setting of a statement attribute.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLGetStmtAttrW(). Refer to Unicode functions (CLI)Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLGetStmtAttr   (SQLHSTMT          StatementHandle,
                              SQLINTEGER        Attribute,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        BufferLength,
                              SQLINTEGER        *StringLengthPtr);
```

Function arguments:

Table 95. SQLGetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLINTEGER	*Attribute*	input	Attribute to retrieve.
SQLPOINTER	*ValuePtr*	output	Pointer to a buffer in which to return the value of the attribute specified in *Attribute*.
SQLINTEGER	*BufferLength*	input	If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. If *Attribute* is an ODBC-defined attribute and **ValuePtr* is an integer, *BufferLength* is ignored. If *Attribute* is a DB2 CLI attribute, the application indicates the nature of the attribute by setting the *BufferLength* argument. *BufferLength* can have the following values: • If **ValuePtr* is a pointer to a character string, then *BufferLength* is the number of bytes needed to store the string, or SQL_NTS. • If **ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *BufferLength*. This places a negative value in *BufferLength*. • If **ValuePtr* is a pointer to a value other than a character string or binary string, then *BufferLength* should have the value SQL_IS_POINTER. • If **ValuePtr* is contains a fixed-length data type, then *BufferLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate. • If the value returned in *ValuePtr* is a Unicode string, the *BufferLength* argument must be an even number.

Table 95. SQLGetStmtAttr arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	*StringLengthPtr*	output	A pointer to a buffer in which to return the total number of bytes (excluding the null termination character) available to return in **ValuePtr*. If this is a null pointer, no length is returned. If the attribute value is a character string, and the number of bytes available to return is greater than or equal to *BufferLength*, the data in **ValuePtr* is truncated to *BufferLength* minus the length of a null termination character and is null-terminated by the DB2 CLI.

Usage:

A call to `SQLGetStmtAttr()` returns in **ValuePtr* the value of the statement attribute specified in *Attribute*. That value can either be a 32-bit value or a null-terminated character string. If the value is a null-terminated string, the application specifies the maximum length of that string in the *BufferLength* argument, and DB2 CLI returns the length of that string in the **StringLengthPtr* buffer. If the value is a 32-bit value, the *BufferLength* and *StringLengthPtr* arguments are not used.

The following statement attributes are read-only, so can be retrieved by `SQLGetStmtAttr()`, but not set by `SQLSetStmtAttr()`. Refer to the list of statement attributes for all statement attributes that can be set and retrieved.
- SQL_ATTR_IMP_PARAM_DESC
- SQL_ATTR_IMP_ROW_DESC
- SQL_ATTR_ROW_NUMBER

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 96. SQLGetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The data returned in **ValuePtr* was truncated to be *BufferLength* minus the length of a null termination character. The length of the untruncated string value is returned in **StringLengthPtr*. (Function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state.	The argument *Attribute* was SQL_ATTR_ROW_NUMBER and the cursor was not open, or the cursor was positioned before the start of the result set or after the end of the result set.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.

Table 96. SQLGetStmtAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	An asynchronously executing function was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()` or `SQLExecDirect()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value specified for argument *BufferLength* was less than 0.
HY092	Option type out of range.	The value specified for the argument *Attribute* was not valid for this version of DB2 CLI
HY109	Invalid cursor position.	The *Attribute* argument was SQL_ATTR_ROW_NUMBER and the row had been deleted or could not be fetched.
HYC00	Driver not capable.	The value specified for the argument *Attribute* was a valid DB2 CLI attribute for the version of DB2 CLI, but was not supported by the data source.

Restrictions:

None.

Example:

```
/* get the handle for the implicitly allocated descriptor */
rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_IMP_ROW_DESC,
                    &hIRD,
                    SQL_IS_INTEGER,
                    &indicator);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLGetConnectAttr function (CLI) - Get current attribute setting" on page 146
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "Statement attributes (CLI) list" on page 348

Related samples:

- "dbinfo.c -- How to get and set information at the database level"
- "dbuse.c -- How to use a database"

SQLGetStmtOption function (CLI) - Return current setting of a statement option

Deprecated:

Note:

In ODBC 3.0, SQLGetStmtOption() has been deprecated and replaced with SQLGetStmtAttr().

Although this version of DB2 CLI continues to support SQLGetStmtOption(), we recommend that you use SQLGetStmtAttr() in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLGetStmtOption(hstmt, SQL_ATTR_CURSOR_HOLD, pvCursorHold);
```

for example, would be rewritten using the new function as:

```
SQLGetStmtAttr(hstmt, SQL_ATTR_CURSOR_HOLD, pvCursorHold,
               SQL_IS_INTEGER, NULL);
```

Related reference:

- "CLI and ODBC function summary" on page 1
- "SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute" on page 216

SQLGetSubString function (CLI) - Retrieve portion of a string value

Purpose:

Specification:	DB2 CLI 2.1		

SQLGetSubString() is used to retrieve a portion of a large object value, referenced by a large object locator that has been returned from the server (returned by a fetch or a previous SQLGetSubString() call) during the current transaction.

Syntax:

```
SQLRETURN  SQLGetSubString   (
             SQLHSTMT          StatementHandle,   /* hstmt */
             SQLSMALLINT       LocatorCType,
             SQLINTEGER        SourceLocator,
             SQLUINTEGER       FromPosition,
             SQLUINTEGER       ForLength,
             SQLSMALLINT       TargetCType,
             SQLPOINTER        DataPtr,           /* rgbValue */
             SQLINTEGER        BufferLength,      /* cbValueMax */
             SQLINTEGER        *StringLength,
             SQLINTEGER        *IndicatorValue);
```

Function arguments:

Table 97. SQLGetSubString arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. This can be any statement handle which has been allocated but which does not currently have a prepared statement assigned to it.
SQLSMALLINT	*LocatorCType*	input	The C type of the source LOB locator. This may be: • SQL_C_BLOB_LOCATOR • SQL_C_CLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR
SQLINTEGER	*Locator*	input	*Locator* must be set to the source LOB locator value.
SQLUINTEGER	*FromPosition*	input	For BLOBs and CLOBs, this is the position of the first byte to be returned by the function. For DBCLOBs, this is the first character. The start byte or character is numbered 1.
SQLUINTEGER	*ForLength*	input	This is the length of the string to be returned by the function. For BLOBs and CLOBs, this is the length in bytes. For DBCLOBs, this is the length in characters. If *FromPosition* is less than the length of the source string but *FromPosition* + *ForLength* - 1 extends beyond the end of the source string, the result is padded on the right with the necessary number of characters (X'00' for BLOBs, single byte blank character for CLOBs, and double byte blank character for DBCLOBs).
SQLSMALLINT	*TargetCType*	input	The C data type of the *DataPtr*. The target must always be either a LOB locator C buffer type: • SQL_C_CLOB_LOCATOR • SQL_C_BLOB_LOCATOR • SQL_C_DBCLOB_LOCATOR or a C string type: • SQL_C_CHAR • SQL_C_WCHAR • SQL_C_BINARY • SQL_C_DBCHAR
SQLPOINTER	*DataPtr*	output	Pointer to the buffer where the retrieved string value or a LOB locator is to be stored.
SQLINTEGER	*BufferLength*	input	Maximum size of the buffer pointed to by *DataPtr* in bytes.
SQLINTEGER *	*StringLength*	output	The length of the returned information in *DataPtr* in bytes[a] if the target C buffer type is intended for a binary or character string variable and not a locator value. If the pointer is set to NULL, nothing is returned.
SQLINTEGER *	*IndicatorValue*	output	Always set to zero.

Note:

a This is in bytes even for DBCLOB data.

Usage:

SQLGetSubString() is used to obtain any portion of the string that is represented by the LOB locator. There are two choices for the target:

- The target can be an appropriate C string variable.
- A new LOB value can be created on the server and the LOB locator for that value can be assigned to a target application variable on the client.

SQLGetSubString() can be used as an alternative to SQLGetData() for getting LOB data in pieces. In this case a column is first bound to a LOB locator, which is then used to fetch the LOB as a whole or in pieces.

The Locator argument can contain any valid LOB locator which has not been explicitly freed using a FREE LOCATOR statement nor implicitly freed because the transaction during which it was created has ended.

The statement handle must not have been associated with any prepared statements or catalog function calls.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 98. SQLGetSubString SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The amount of data to be returned is longer than *BufferLength*. The actual length of data available for return is stored in *StringLength*.
07006	Invalid conversion.	The value specified for *TargetCType* was not SQL_C_CHAR, SQL_WCHAR, SQL_C_BINARY, SQL_C_DBCHAR, or a LOB locator. The value specified for *TargetCType* is inappropriate for the source (for example SQL_C_DBCHAR for a BLOB column).
22011	A substring error occurred.	*FromPosition* is greater than the of length of the source string.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY003	Program type out of range.	*LocatorCType* is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY009	Invalid argument value.	The value specified for *FromPosition* or for *ForLength* was not a positive integer.

Table 98. SQLGetSubString SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	The specified *StatementHandle* is not in an *allocated* state. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The value of *BufferLength* was less than 0.
HYC00	Driver not capable.	The application is currently connected to a data source that does not support large objects.
0F001	No locator currently assigned	The value specified for *Locator* is not currently a LOB locator.

Restrictions:

This function is not available when connected to a DB2 server that does not support large objects. Call `SQLGetFunctions()` with the function type set to SQL_API_SQLGETSUBSTRING and check the *fExists* output argument to determine if the function is supported for the current connection.

Example:

```
/* read the piece of CLOB data in buffer */
cliRC = SQLGetSubString(hstmtLocUse,
                        SQL_C_CLOB_LOCATOR,
                        clobLoc,
                        clobPiecePos,
                        clobLen - clobPiecePos,
                        SQL_C_CHAR,
                        buffer,
                        clobLen - clobPiecePos + 1,
                        &clobPieceLen,
                        &ind);
```

Related concepts:

- "LOB usage in ODBC applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126
- "SQLGetData function (CLI) - Get data from a column" on page 152
- "SQLGetLength function (CLI) - Retrieve length of a string value" on page 210

- "SQLGetPosition function (CLI) - Return starting position of string" on page 213

Related samples:

- "spserver.c -- Definition of various types of stored procedures"

SQLGetTypeInfo function (CLI) - Get data type information

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLGetTypeInfo()` returns information about the data types that are supported by the DBMSs associated with DB2 CLI. The information is returned in an SQL result set. The columns can be received using the same functions that are used to process a query.

Syntax:

```
SQLRETURN   SQLGetTypeInfo (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLSMALLINT       DataType);        /* fSqlType */
```

Function arguments:

Table 99. SQLGetTypeInfo arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.

Table 99. SQLGetTypeInfo arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*DataType*	input	The SQL data type being queried. The supported types are: • SQL_ALL_TYPES • SQL_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_CHAR • SQL_CLOB • SQL_DATE • SQL_DBCLOB • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TIME • SQL_TIMESTAMP • SQL_TINYINT • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_XML If SQL_ALL_TYPES is specified, information about all supported data types would be returned in ascending order by TYPE_NAME. All unsupported data types would be absent from the result set.

Usage:

Since `SQLGetTypeInfo()` generates a result set and is equivalent to executing a query, it will generate a cursor and begin a transaction. To prepare and execute another statement on this statement handle, the cursor must be closed.

If `SQLGetTypeInfo()` is called with an invalid *DataType*, an empty result set is returned.

If either the LONGDATACOMPAT keyword or the SQL_ATTR_LONGDATA_COMPAT connection attribute is set, then SQL_LONGVARBINARY, SQL_LONGVARCHAR and SQL_LONGVARGRAPHIC will be returned for the *DATA_TYPE* argument instead of SQL_BLOB, SQL_CLOB and SQL_DBCLOB.

The columns of the result set generated by this function are described below.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change. The data types returned are those that can be used in a CREATE TABLE, ALTER

TABLE, DDL statement. Non-persistent data types such as the locator data types are not part of the returned result set. User-defined data types are not returned either.

Columns returned by SQLGetTypeInfo

Column 1 TYPE_NAME (VARCHAR(128) NOT NULL Data Type)
Data source-dependent data type name; for example, ″CHAR()″, ″LONG VARBINARY″. Applications must use this name in the CREATE TABLE and ALTER TABLE statements.

Column 2 DATA_TYPE (SMALLINT NOT NULL Data Type)
SQL data type define values, for example, SQL_VARCHAR, SQL_BLOB, SQL_DATE, SQL_INTEGER.

Column 3 COLUMN_SIZE (INTEGER Data Type)
If the data type is a character or binary string, then this column contains the maximum length in bytes; if it is a graphic (DBCS) string, this is the number of double byte characters for the column (the CLI/ODBC configuration keyword Graphic can change this default behaviour). If the data type is XML, zero is returned.

For date, time, timestamp data types, this is the total number of characters required to display the value when converted to character.

For numeric data types, this is the total number of digits (precision).

Column 4 LITERAL_PREFIX (VARCHAR(128) Data Type)
Character that DB2 recognizes as a prefix for a literal of this data type. This column is null for data types where a literal prefix is not applicable.

Column 5 LITERAL_SUFFIX (VARCHAR(128) Data Type)
Character that DB2 recognizes as a suffix for a literal of this data type. This column is null for data types where a literal prefix is not applicable.

Column 6 CREATE_PARAMS (VARCHAR(128) Data Type)
The text of this column contains a list of keywords, separated by commas, corresponding to each parameter the application might specify in parenthesis when using the name in the TYPE_NAME column as a data type in SQL. The keywords in the list can be any of the following: LENGTH, PRECISION, SCALE. They appear in the order that the SQL syntax requires that they be used.

A NULL indicator is returned if there are no parameters for the data type definition, (such as INTEGER).

Note: The intent of CREATE_PARAMS is to enable an application to customize the interface for a *DDL builder*. An application should expect, using this, only to be able to determine the number of arguments required to define the data type and to have localized text that could be used to label an edit control.

Column 7 NULLABLE (SMALLINT NOT NULL Data Type)
Indicates whether the data type accepts a NULL value
- Set to SQL_NO_NULLS if NULL values are disallowed.
- Set to SQL_NULLABLE if NULL values are allowed.
- Set to SQL_NULLABLE_UNKNOWN if it is not known whether NULL values are allowed or not.

Column 8 CASE_SENSITIVE (SMALLINT NOT NULL Data Type)
Indicates whether a character data type is case-sensitive in collations and comparisons. Valid values are SQL_TRUE and SQL_FALSE.

Column 9 SEARCHABLE (SMALLINT NOT NULL Data Type)
Indicates how the data type is used in a WHERE clause. Valid values are:
- SQL_UNSEARCHABLE : if the data type cannot be used in a WHERE clause.
- SQL_LIKE_ONLY : if the data type can be used in a WHERE clause only with the LIKE predicate.
- SQL_ALL_EXCEPT_LIKE : if the data type can be used in a WHERE clause with all comparison operators except LIKE.
- SQL_SEARCHABLE : if the data type can be used in a WHERE clause with any comparison operator.

Column 10 UNSIGNED_ATTRIBUTE (SMALLINT Data Type)
Indicates whether the data type is unsigned. The valid values are: SQL_TRUE, SQL_FALSE or NULL. A NULL indicator is returned if this attribute is not applicable to the data type.

Column 11 FIXED_PREC_SCALE (SMALLINT NOT NULL Data Type)
Contains the value SQL_TRUE if the data type is exact numeric and always has the same precision and scale; otherwise, it contains SQL_FALSE.

Column 12 AUTO_INCREMENT (SMALLINT Data Type)
Contains SQL_TRUE if a column of this data type is automatically set to a unique value when a row is inserted; otherwise, contains SQL_FALSE.

Column 13 LOCAL_TYPE_NAME (VARCHAR(128) Data Type)
This column contains any localized (native language) name for the data type that is different from the regular name of the data type. If there is no localized name, this column is NULL.

This column is intended for display only. The character set of the string is locale-dependent and is typically the default character set of the database.

Column 14 MINIMUM_SCALE (INTEGER Data Type)
The minimum scale of the SQL data type. If a data type has a fixed scale, the MINIMUM_SCALE and MAXIMUM_SCALE columns both contain the same value. NULL is returned where scale is not applicable.

Column 15 MAXIMUM_SCALE (INTEGER Data Type)
The maximum scale of the SQL data type. NULL is returned where scale is not applicable. If the maximum scale is not defined separately in the DBMS, but is defined instead to be the same as the maximum length of the column, then this column contains the same value as the COLUMN_SIZE column.

Column 16 SQL_DATA_TYPE (SMALLINT NOT NULL Data Type)
The value of the SQL data type as it appears in the SQL_DESC_TYPE field of the descriptor. This column is the same as the DATA_TYPE column (except for interval and datetime data types which DB2 CLI does not support).

Column 17 SQL_DATETIME_SUB (SMALLINT Data Type)
This field is always NULL (DB2 CLI does not support interval and datetime data types).

Column 18 NUM_PREC_RADIX (INTEGER Data Type)
If the data type is an approximate numeric type, this column contains the

value 2 to indicate that COLUMN_SIZE specifies a number of bits. For exact numeric types, this column contains the value 10 to indicate that COLUMN_SIZE specifies a number of decimal digits. Otherwise, this column is NULL.

Column 19 INTERVAL_PRECISION (SMALLINT Data Type)
This field is always NULL (DB2 CLI does not support interval data types).

Return codes:
- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 100. SQLGetTypeInfo SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle. *StatementHandle* had not been closed.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY004	SQL data type out of range.	An invalid *DataType* was specified.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Example:

```
/* get data type information */
cliRC = SQLGetTypeInfo(hstmt, SQL_ALL_TYPES);
```

Related concepts:
- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLFetchScroll function (CLI) - Fetch rowset and return data for all bound columns" on page 126

- "SQLGetInfo function (CLI) - Get general information" on page 180
- "SQLSetColAttributes function (CLI) - Set column attributes" on page 266
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*
- "ALTER TABLE statement" in *SQL Reference, Volume 2*
- "CREATE TABLE statement" in *SQL Reference, Volume 2*

Related samples:

- "dtinfo.c -- How get information about data types"

SQLMoreResults function (CLI) - Determine if there are more result sets

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

`SQLMoreResults()` determines whether there is more information available on the statement handle which has been associated with:

- an array input of parameter values for a query
- a stored procedure that is returning result sets
- or batched SQL

Syntax:

```
SQLRETURN   SQLMoreResults   (SQLHSTMT     StatementHandle);  /* hstmt */
```

Function arguments:

Table 101. SQLMoreResults arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.

Usage:

This function is used to return multiple results set in a sequential manner upon the execution of:

- a parameterized query with an array of input parameter values specified with the SQL_ATTR_PARAMSET_SIZE statement attribute and `SQLBindParameter()`, or
- a stored procedure containing SQL queries, the cursors of which have been left open so that the result sets remain accessible when the stored procedure has finished execution. For this scenario, the stored procedure is typically trying to return multiple result sets.
- or batched SQL. When multiple SQL statements are batched together during a single `SQLExecute()` or `SQLExecDirect()`.

After completely processing the first result set, the application can call `SQLMoreResults()` to determine if another result set is available. If the current result set has unfetched rows, `SQLMoreResults()` discards them by closing the cursor and, if another result set is available, returns SQL_SUCCESS.

If all the result sets have been processed, SQLMoreResults() returns SQL_NO_DATA_FOUND.

Applications that want to be able to manipulate more than one result set at the same time can use the DB2 CLI function SQLNextResult() to move a result set to another statement handle. SQLNextResult() does not support batched statements.

When using batched SQL, SQLExecute() or SQLExecDirect() will only execute the first SQL statement in the batch. SQLMoreResults() can then be called to execute the next SQL statement and will return SQL_SUCCESS if the next statement is successfully executed. If there are no more statements to be executed, then SQL_NO_DATA_FOUND is returned. If the batched SQL statement is an UPDATE, INSERT, or DELETE statement, then SQLRowCount() can be called to determine the number of rows affected.

If SQLCloseCursor() or if SQLFreeStmt() is called with the SQL_CLOSE option, or SQLFreeHandle() is called with *HandleType* set to SQL_HANDLE_STMT, all pending result sets on this statement handle are discarded.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

Diagnostics:

Table 102. SQLMoreResults SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

In addition SQLMoreResults() can return the SQLSTATEs associated with SQLExecute().

Example:

```
cliRC = SQLMoreResults(hstmt);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Calling stored procedures from CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLFreeHandle function (CLI) - Free handle resources" on page 140
- "SQLFreeStmt function (CLI) - Free (or reset) a statement handle" on page 143
- "SQLNextResult function (CLI) - Associate next result set with another statement handle" on page 235
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spclient.c -- Call various stored procedures"
- "spcall.c -- Call individual stored procedures"
- "spclires.c -- Contrast stored procedure multiple result set handling methods"

SQLNativeSql function (CLI) - Get native SQL text

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLNativeSql()` is used to show how DB2 CLI interprets vendor escape clauses. If the original SQL string passed in by the application contained vendor escape clause sequences, then DB2 CLI will return the transformed SQL string that would be seen by the data source (with vendor escape clauses either converted or discarded, as appropriate).

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLNativeSqlW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLNativeSql     (
               SQLHDBC           ConnectionHandle,  /* hdbc */
               SQLCHAR           *InStatementText,  /* szSqlStrIn */
               SQLINTEGER        TextLength1,       /* cbSqlStrIn */
               SQLCHAR           *OutStatementText, /* szSqlStr */
               SQLINTEGER        BufferLength,      /* cbSqlStrMax */
               SQLINTEGER        *TextLength2Ptr);  /* pcbSqlStr */
```

Function arguments:

Table 103. SQLNativeSql arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection Handle
SQLCHAR *	*InStatementText*	input	Input SQL string
SQLINTEGER	*TextLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *InStatementText*.
SQLCHAR *	*OutStatementText*	output	Pointer to buffer for the transformed output string
SQLINTEGER	*BufferLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *OutStatementText*.
SQLINTEGER *	*TextLength2Ptr*	output	The total number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-terminator, available to return in *OutStatementText*. If the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) available to return is greater than or equal to *BufferLength*, the output SQL string in *OutStatementText* is truncated to *BufferLength* - 1 SQLCHAR or SQLWCHAR elements.

Usage:

This function is called when the application wishes to examine or display the transformed SQL string that would be passed to the data source by DB2 CLI. Translation (mapping) would only occur if the input SQL statement string contains vendor escape clause sequence(s).

DB2 CLI can only detect vendor escape clause syntax errors when `SQLNativeSql()` is called. Because DB2 CLI does not pass the transformed SQL string to the data source for preparation, syntax errors that are detected by the DBMS are not generated at this time. (The statement is not passed to the data source for preparation because the preparation may potentially cause the initiation of a transaction.)

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 104. SQLNativeSql SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The buffer *OutStatementText* was not large enough to contain the entire SQL string, so truncation occurred. The argument *TextLength2Ptr* contains the total length of the untruncated SQL string. (Function returns with SQL_SUCCESS_WITH_INFO)
08003	Connection is closed.	The *ConnectionHandle* does not reference an open database connection.
37000	Invalid SQL syntax.	The input SQL string in *InStatementText* contained a syntax error.

Table 104. SQLNativeSql SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	The argument *InStatementText* is a NULL pointer. The argument *OutStatementText* is a NULL pointer.
HY090	Invalid string or buffer length.	The argument *TextLength1* was less than 0, but not equal to SQL_NTS. The argument *BufferLength* was less than 0.

Restrictions:

None.

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Vendor escape clauses in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLNumParams function (CLI) - Get number of parameters in a SQL statement

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

`SQLNumParams()` returns the number of parameter markers in an SQL statement.

Syntax:

```
SQLRETURN   SQLNumParams (
               SQLHSTMT         StatementHandle,      /* hstmt */
               SQLSMALLINT      *ParameterCountPtr);  /* pcpar */
```

Function arguments:

Table 105. SQLNumParams arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLSMALLINT *	*ParameterCountPtr*	Output	Number of parameters in the statement.

Usage:

If the prepared SQL statement associated with *Statement Handle* contains batch SQL (multiple SQL statements separated by a semicolon ';'), the parameters are counted for the entire string and are not differentiated by the individual statements making up the batch.

This function can only be called after the statement associated with *StatementHandle* has been prepared. If the statement does not contain any parameter markers, *ParameterCountPtr* is set to 0.

An application can call this function to determine how many `SQLBindParameter()` (or `SQLBindFileToParam()`) calls are necessary for the SQL statement associated with the statement handle.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 106. SQLNumParams SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	This function was called before `SQLPrepare()` was called for the specified *StatementHandle* The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Parameter marker binding in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

SQLNextResult function (CLI) - Associate next result set with another statement handle

Purpose:

Specification:	DB2 CLI 7.x		

`SQLNextResult()` allows non-sequential access to multiple result sets returned from a stored procedure.

Syntax:

```
SQLRETURN   SQLNextResult (SQLHSTMT StatementHandle1
                           SQLHSTMT StatementHandle2);
```

Function arguments:

Table 107. SQLNextResult arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle1*	input	Statement handle.
SQLHSTMT	*StatementHandle2*	input	Statement handle.

Usage:

A stored procedure returns multiple result sets by leaving one or more cursors open after exiting. The first result set is always accessed by using the statement handle that called the stored procedure. If multiple result sets are returned, either `SQLMoreResults()` or `SQLNextResult()` can be used to describe and fetch the result set.

`SQLMoreResults()` is used to close the cursor for the first result set and allow the next result set to be processed on the same statement handle, whereas `SQLNextResult()` moves the next result set to *StatementHandle2*, without closing the cursor on *StatementHandle1*. Both functions return SQL_NO_DATA_FOUND if there are no result sets to be fetched.

Using `SQLNextResult()` allows result sets to be processed in any order once they have been transferred to other statement handles. Mixed calls to `SQLMoreResults()` and `SQLNextResult()` are allowed until there are no more cursors (open result sets) on *StatementHandle1*.

When `SQLNextResult()` returns SQL_SUCCESS, the next result set is no longer associated with *StatementHandle1*. Instead, the next result set is associated with *StatementHandle2*, as if a call to `SQLExecDirect()` had just successfully executed a query on *StatementHandle2*. The cursor, therefore, can be described using `SQLNumResultCols()`, `SQLDescribeCol()`, or `SQLColAttribute()`.

After `SQLNextResult()` has been called, the result set now associated with *StatementHandle2* is removed from the chain of remaining result sets and cannot be

used again in either SQLNextResult() or SQLMoreResults(). This means that for 'n' result sets, SQLNextResult() can be called successfully at most 'n-1' times.

If SQLCloseCursor() or if SQLFreeStmt() is called with the SQL_CLOSE option, or SQLFreeHandle() is called with *HandleType* set to SQL_HANDLE_STMT, all pending result sets on this statement handle are discarded.

SQLNextResult() returns SQL_ERROR if *StatementHandle2* has an open cursor or *StatementHandle1* and *StatementHandle2* are not on the same connection. If any errors or warnings are returned, SQLGetDiagRec() must always be called on *StatementHandle1*.

Note: SQLMoreResults() also works with a parameterized query with an array of input parameter values specified with the SQL_ATTR_ROW_ARRAY_SIZE statement attribute and SQLBindParameter(). SQLNextResult(), however, does not support this.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NO_DATA_FOUND

Diagnostics:

Table 108. SQLNextResult SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication Link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate the memory required to support execution or completion of the function.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation. *StatementHandle2* has an open cursor associated with it. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access the memory required to support execution or completion of the function.
HYT00	Time-out expired.	The time-out period expired before the data source returned the result set. The time-out period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for SQLSetStmtAttr().

Restrictions:

Only SQLMoreResults() can be used for parameterized queries and batched SQL.

Example:

```
/* use SQLNextResult to push Result Set 2 onto the second statement handle */
cliRC = SQLNextResult(hstmt, hstmt2); /* open second cursor */
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Calling stored procedures from CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLMoreResults function (CLI) - Determine if there are more result sets" on page 229
- "Statement attributes (CLI) list" on page 348

Related samples:

- "spclires.c -- Contrast stored procedure multiple result set handling methods"

SQLNumResultCols function (CLI) - Get number of result columns

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLNumResultCols()` returns the number of columns in the result set associated with the input statement handle.

`SQLPrepare()` or `SQLExecDirect()` must be called before calling this function.

After calling this function, you can call `SQLColAttribute()`, or one of the bind column functions.

Syntax:

```
SQLRETURN   SQLNumResultCols (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLSMALLINT       *ColumnCountPtr);  /* pccol */
```

Function arguments:

Table 109. SQLNumResultCols arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLSMALLINT *	*ColumnCountPtr*	output	Number of columns in the result set

Usage:

The function sets the output argument to zero if the last statement or function executed on the input statement handle did not generate a result set.

Return codes:

- SQL_SUCCESS

- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 110. SQLNumResultCols SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called prior to calling `SQLPrepare()` or `SQLExecDirect()` for the *StatementHandle*. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Authorization:

None.

Example:

```
/* identify the number of output columns */
cliRC = SQLNumResultCols(hstmt, &nResultCols);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Result set terminology in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10

- "SQLBindFileToCol function (CLI) - Bind LOB file reference to LOB column" on page 16
- "SQLDescribeCol function (CLI) - Return a set of attributes for a column" on page 82
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLPrepare function (CLI) - Prepare a statement" on page 242
- "SQLSetColAttributes function (CLI) - Set column attributes" on page 266
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "spclient.c -- Call various stored procedures"
- "spclires.c -- Contrast stored procedure multiple result set handling methods"
- "tbread.c -- How to read data from tables"
- "dbuse.c -- How to use a database"

SQLParamData function (CLI) - Get next parameter for which a data value is needed

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

`SQLParamData()` is used in conjunction with `SQLPutData()` to send long data in pieces. It can also be used to send fixed-length data at execution time.

Syntax:

```
SQLRETURN   SQLParamData (
               SQLHSTMT         StatementHandle, /* hstmt */
               SQLPOINTER       *ValuePtrPtr );  /* prgbValue */
```

Function arguments:

Table 111. SQLParamData arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLPOINTER *	*ValuePtrPtr*	output	Pointer to a buffer in which to return the address of the *ParameterValuePtr* buffer specified in `SQLBindParameter()` (for parameter data) or the address of the *TargetValuePtr* buffer specified in `SQLBindCol()` (for column data), as contained in the SQL_DESC_DATA_PTR descriptor record field.

Usage:

`SQLParamData()` returns SQL_NEED_DATA if there is at least one SQL_DATA_AT_EXEC parameter for which data still has not been assigned. This function returns an application-provided value in *ValuePtrPtr* supplied by the application during a previous `SQLBindParameter()` call. `SQLPutData()` is called one or more times (in the case of long data) to send the parameter data. `SQLParamData()` is called to signal that all the data has been sent for the current parameter and to advance to the next SQL_DATA_AT_EXEC parameter.

SQL_SUCCESS is returned when all the parameters have been assigned data values and the associated statement has been executed successfully. If any errors occur during or before actual statement execution, SQL_ERROR is returned.

If `SQLParamData()` returns SQL_NEED_DATA, then only `SQLPutData()` or `SQLCancel()` calls can be made. All other function calls using this statement handle will fail. In addition, all function calls referencing the parent connection handle of *StatementHandle* will fail if they involve changing any attribute or state of that connection; that is, that following function calls on the parent connection handle are also not permitted:

- SQLSetConnectAttr()
- SQLEndTran()

Should they be invoked during an SQL_NEED_DATA sequence, these functions will return SQL_ERROR with SQLSTATE of **HY**010 and the processing of the SQL_DATA_AT_EXEC parameters will not be affected.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE
- SQL_NEED_DATA

Diagnostics:

`SQLParamData()` can return any SQLSTATE returned by the `SQLPrepare()`, `SQLExecDirect()`, and `SQLExecute()` functions. In addition, the following diagnostics can also be generated:

Table 112. SQLParamData SQLSTATEs

SQLSTATE	Description	Explanation
07006	Invalid conversion.	Transfer of data between DB2 CLI and the application variables would result in incompatible data conversion.
22026	String data, length mismatch	The SQL_NEED_LONG_DATA_LEN information type in `SQLGetInfo()` was 'Y' and less data was sent for a long parameter (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or other long data type) than was specified with the *StrLen_or_IndPtr* argument in `SQLBindParameter()`. The SQL_NEED_LONG_DATA_LEN information type in `SQLGetInfo()` was 'Y' and less data was sent for a long column (the data type was SQL_LONGVARCHAR, SQL_LONGVARBINARY, or other long data type) than was specified in the length buffer corresponding to a column in a row of data that was updated with `SQLSetPos()`.
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to a deadlock or timeout.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.

Table 112. SQLParamData SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by `SQLGetDiagRec()` in the argument *MessageText* describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	`SQLParamData()` was called out of sequence. This call is only valid after an `SQLExecDirect()` or an `SQLExecute()`, or after an `SQLPutData()` call. Even though this function was called after an `SQLExecDirect()` or an `SQLExecute()` call, there were no SQL_DATA_AT_EXEC parameters (left) to process.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY092	Option type out of range.	The *FileOptions* argument of a previous `SQLBindFileToParam()` operation was not valid.
HY506	Error closing a file.	Error encountered while trying to close a temporary file.
HY509	Error deleting a file.	Error encountered while trying to delete a temporary file.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get next parameter for which a data value is needed */
cliRC = SQLParamData(hstmt, (SQLPOINTER *)&valuePtr);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Large object usage in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Long data for bulk inserts and updates in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23

- "SQLCancel function (CLI) - Cancel statement" on page 49
- "SQLPutData function (CLI) - Passing data value for a parameter" on page 261
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:
- "dtlob.c -- How to read and write LOB data"

SQLParamOptions function (CLI) - Specify an input array for a parameter

Deprecated:

Note:

In ODBC 3.0, `SQLParamOptions()` has been deprecated and replaced with `SQLSetStmtAttr()`.

Although this version of DB2 CLI continues to support `SQLParamOptions()`, we recommend that you use `SQLSetStmtAttr()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLParamOptions(hstmt, crow, pirow);
```

for example, would be rewritten using the new function as:

```
SQLSetStmtAttr(hstmt, fOption, pvParam, fStrLen);
```

Related reference:
- "CLI and ODBC function summary" on page 1
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294

SQLPrepare function (CLI) - Prepare a statement

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

`SQLPrepare()` associates an SQL statement or XQuery expression with the input statement handle provided. The application can include one or more parameter markers in the SQL statement. To include a parameter marker, the application embeds a question mark (?) into the SQL string at the appropriate position. The application can reference this prepared statement by passing the statement handle to other functions.

Note: For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the XMLQUERY function to bind parameter markers to XQuery variables. The values of the bound parameter markers will then be passed to the XQuery expression specified in XMLQUERY for execution.

If the statement handle has been previously used with a query statement (or any function that returns a result set), either SQLCloseCursor() or SQLFreeStmt() with the SQL_CLOSE option must be called to close the cursor before calling SQLPrepare().

XQuery expressions must be prefixed with the "XQUERY" keyword. To prepare and execute XQuery expressions without having to include this keyword, set the statement attribute SQL_ATTR_XQUERY_STATEMENT to SQL_TRUE before calling SQLPrepare() or SQLExecDirect().

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLPrepareW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLPrepare (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLCHAR           *StatementText,   /* szSqlStr */
               SQLINTEGER        TextLength);      /* cbSqlStr */
```

Function arguments:

Table 113. SQLPrepare arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle. There must not be an open cursor associated with *StatementHandle*.
SQLCHAR *	*StatementText*	input	SQL statement string
SQLINTEGER	*TextLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *StatementText* argument, or SQL_NTS if *StatementText* is null-terminated.

Usage:

Deferred prepare is on by default. The prepare request is not sent to the server until either SQLDescribeParam(), SQLExecute(), SQLNumResultCols(), SQLDescribeCol(), or SQLColAttribute() is called using the same statement handle as the prepared statement. This minimizes network flow and improves performance.

If the SQL statement text contains vendor escape clause sequences, DB2 CLI will first modify the SQL statement text to the appropriate DB2 specific format before submitting it to the database for preparation. If the application does not generate SQL statements that contain vendor escape clause sequences then the SQL_ATTR_NOSCAN statement attribute should be set to SQL_NOSCAN at the connection level so that DB2 CLI does not perform a scan for any vendor escape clauses.

Once a statement has been prepared using SQLPrepare(), the application can request information about the format of the result set (if the statement was a query) by calling:

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttribute()

Information about the parameter markers in *StatementText* can be requested using the following:

- `SQLDescribeParam()`
- `SQLNumParams()`

Note: The first invocation of any of the above functions except `SQLNumParams()` will force the PREPARE request to be sent to the server if deferred prepare is enabled.

The SQL statement string might contain parameter markers and `SQLNumParams()` can be called to determine the number of parameter markers in the statement. A parameter marker is represented by a "?" character, and is used to indicate a position in the statement where an application-supplied value is to be substituted when `SQLExecute()` is called. The bind parameter functions, `SQLBindParameter()`, `SQLSetParam()` and `SQLBindFileToParam()`, are used to bind or associate application variables with each parameter marker and to indicate if any data conversion should be performed at the time the data is transferred. An application can call `SQLDescribeParam()` to retrieve information about the data expected by the database server for the parameter marker.

All parameters must be bound before calling `SQLExecute()`.

Refer to the PREPARE statement for information on rules related to parameter markers.

Once the application has processed the results from the `SQLExecute()` call, it can execute the statement again with new (or the same) parameter values.

The SQL statement can be COMMIT or ROLLBACK and executing either of these statements has the same effect as calling `SQLEndTran()` on the current connection handle.

If the SQL statement is a positioned DELETE or a positioned UPDATE, the cursor referenced by the statement must be defined on a separate statement handle under the same connection handle and same isolation level.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 114. SQLPrepare SQLSTATEs

SQLSTATE	Description	Explanation
01504	The UPDATE or DELETE statement does not include a WHERE clause.	*StatementText* contained an UPDATE or DELETE statement which did not contain a WHERE clause.
01508	Statement disqualified for blocking.	The statement was disqualified for blocking for reasons other than storage.
21S01	Insert value list does not match column list.	*StatementText* contained an INSERT statement and the number of values to be inserted did not match the degree of the derived table.

Table 114. SQLPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
21S02	Degrees of derived table does not match column list.	*StatementText* contained a CREATE VIEW statement and the number of names specified is not the same degree as the derived table defined by the query specification.
22018	Invalid character value for cast specification.	*StatementText* contained an SQL statement that contained a literal or parameter and the value was incompatible with the data type of the associated table column.
22019	Invalid escape character	The argument *StatementText* contained a LIKE predicate with an ESCAPE in the WHERE clause, and the length of the escape character following ESCAPE was not equal to 1.
22025	Invalid escape sequence	The argument *StatementText* contained "LIKE *pattern value* ESCAPE *escape character*" in the WHERE clause, and the character following the escape character in the pattern value was not one of "%" or "_".
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
34000	Invalid cursor name.	*StatementText* contained a positioned DELETE or a positioned UPDATE and the cursor referenced by the statement being executed was not open.
37xxx [a]	Invalid SQL syntax.	*StatementText* contained one or more of the following: • an SQL statement that the connected database server could not prepare • a statement containing a syntax error
40001	Transaction rollback.	The transaction to which this SQL statement belonged was rolled back due to deadlock or timeout.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42xxx [a]	Syntax Error or Access Rule Violation.	**42**5xx indicates the authorization ID does not have permission to execute the SQL statement contained in *StatementText*. Other **42**xxx SQLSTATES indicate a variety of syntax or access problems with the statement.
58004	Unexpected system failure.	Unrecoverable system error.
S0001	Database object already exists.	*StatementText* contained a CREATE TABLE or CREATE VIEW statement and the table name or view name specified already existed.
S0002	Database object does not exist.	*StatementText* contained an SQL statement that references a table name or a view name which did not exist.
S0011	Index already exists.	*StatementText* contained a CREATE INDEX statement and the specified index name already existed.
S0012	Index not found.	*StatementText* contained a DROP INDEX statement and the specified index name did not exist.
S0021	Column already exists.	*StatementText* contained an ALTER TABLE statement and the column specified in the ADD clause was not unique or identified an existing column in the base table.
S0022	Column not found.	*StatementText* contained an SQL statement that references a column name which did not exist.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

Table 114. SQLPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	*StatementText* was a null pointer.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The argument *TextLength* was less than 1, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Note:

a xxx refers to any SQLSTATE with that class code. Example, **37**xxx refers to any SQLSTATE in the **37** class.

Note: Not all DBMSs report all of the above diagnostic messages at prepare time. If deferred prepare is left on as the default behavior (controlled by the SQL_ATTR_DEFERRED_PREPARE statement attribute), then these errors could occur when the PREPARE is flowed to the server. The application must be able to handle these conditions when calling functions that cause this flow. These functions include `SQLExecute()`, `SQLDescribeParam()`, `SQLNumResultCols()`, `SQLDescribeCol()`, and `SQLColAttribute()`.

Authorization:

None.

Example:

```
SQLCHAR *stmt = (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? ";

/* ... */

/* prepare the statement */
cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Preparing and executing SQL statements in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBindFileToParam function (CLI) - Bind LOB file reference to LOB parameter" on page 20
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLColAttribute function (CLI) - Return a column attribute" on page 54
- "SQLDescribeCol function (CLI) - Return a set of attributes for a column" on page 82
- "SQLDescribeParam function (CLI) - Return description of a parameter marker" on page 86
- "SQLEndTran function (CLI) - End transactions of a connection or an Environment" on page 97
- "SQLExecute function (CLI) - Execute a statement" on page 106
- "SQLNumParams function (CLI) - Get number of parameters in a SQL statement" on page 233
- "COMMIT statement" in *SQL Reference, Volume 2*
- "PREPARE statement" in *SQL Reference, Volume 2*
- "ROLLBACK statement" in *SQL Reference, Volume 2*
- "SQLNumResultCols function (CLI) - Get number of result columns" on page 237
- "SQLSetParam function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 286
- "Statement attributes (CLI) list" on page 348

Related samples:

- "spserver.c -- Definition of various types of stored procedures"
- "tbmod.c -- How to modify table data"
- "dbuse.c -- How to use a database"

SQLPrimaryKeys function (CLI) - Get primary key columns of a table

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLPrimaryKeys()` returns a list of column names that comprise the primary key for a table. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLPrimaryKeysW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLPrimaryKeys   (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR           *SchemaName,       /* szSchemaName */
               SQLSMALLINT       NameLength2,       /* cbSchemaName */
               SQLCHAR           *TableName,        /* szTableName */
               SQLSMALLINT       NameLength3);      /* cbTableName */
```

Function arguments:

Table 115. SQLPrimaryKeys arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR *	*CatalogName*	input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	input	Schema qualifier of table name.
SQLSMALLINT	*NameLength2*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*TableName*	input	Table name.
SQLSMALLINT	*NameLength3*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.

Usage:

`SQLPrimaryKeys()` returns the primary key columns from a single table. Search patterns cannot be used to specify any of the arguments.

The result set contains the columns listed in "Columns Returned By SQLPrimaryKeys" on page 249, ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME and ORDINAL_POSITION.

Since calls to `SQLPrimaryKeys()` in many cases map to a complex and, thus, expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

If the schema name is not provided, then the schema name defaults to the one currently in effect for the current connection.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively,

call SQLGetInfo() with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns Returned By SQLPrimaryKeys

Column 1 TABLE_CAT (VARCHAR(128))
Primary key table catalog name. The value is NULL if this table does not have catalogs.

Column 2 TABLE_SCHEM (VARCHAR(128))
The name of the schema containing TABLE_NAME.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the specified table.

Column 4 COLUMN_NAME (VARCHAR(128) not NULL)
Primary key column name.

Column 5 KEY_SEQ (SMALLINT not NULL)
Column sequence number in the primary key, starting with 1.

Column 6 PK_NAME (VARCHAR(128))
Primary key identifier. NULL if not applicable to the data source.

Note: The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents and order are identical to those defined for the SQLPrimaryKeys() result set in ODBC.

If the specified table does not contain a primary key, an empty result set is returned.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 116. SQLPrimaryKeys SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

Table 116. SQLPrimaryKeys SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
HYC00	Driver not capable.	DB2 CLI does not support *catalog* as a qualifier for table name.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get the primary key columns of a table */
cliRC = SQLPrimaryKeys(hstmt, NULL, 0, tbSchema, SQL_NTS, tbName, SQL_NTS);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Primary keys" in *Administration Guide: Planning*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLForeignKeys function (CLI) - Get the list of foreign key columns" on page 134
- "SQLStatistics function (CLI) - Get index and statistics information for a base table" on page 305

Related samples:

- "tbconstr.c -- How to work with constraints associated with tables"

SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

SQLProcedureColumns() returns a list of input and output parameters associated with a stored procedure. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLProcedureColumnsW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN SQLProcedureColumns(
             SQLHSTMT         StatementHandle,   /* hstmt */
             SQLCHAR          *CatalogName,      /* szProcCatalog */
             SQLSMALLINT      NameLength1,       /* cbProcCatalog */
             SQLCHAR          *SchemaName,       /* szProcSchema */
             SQLSMALLINT      NameLength2,       /* cbProcSchema */
             SQLCHAR          *ProcName,         /* szProcName */
             SQLSMALLINT      NameLength3,       /* cbProcName */
             SQLCHAR          *ColumnName,       /* szColumnName */
             SQLSMALLINT      NameLength4);      /* cbColumnName */
```

Function arguments:

Table 117. SQLProcedureColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLCHAR *	*CatalogName*	input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	input	Buffer that can contain a *pattern value* to qualify the result set by schema name. For DB2 for MVS/ESA V 4.1 and above, all the stored procedures are in one schema; the only acceptable value for the *SchemaName* argument is a null pointer. If a value is specified, an empty result set and SQL_SUCCESS are returned. For DB2 Database for Linux, UNIX, and Windows, *SchemaName* can contain a valid pattern value. For more information about valid search patterns, refer to the catalog functions input arguments.

Table 117. SQLProcedureColumns arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*NameLength2*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*ProcName*	input	Buffer that can contain a *pattern value* to qualify the result set by procedure name.
SQLSMALLINT	*NameLength3*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *ProcName*, or SQL_NTS if *ProcName* is null-terminated.
SQLCHAR *	*ColumnName*	input	Buffer that can contain a *pattern value* to qualify the result set by parameter name. This argument is to be used to further qualify the result set already restricted by specifying a non-empty value for ProcName, SchemaName, or both.
SQLSMALLINT	*NameLength4*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *ColumnName*, or SQL_NTS if *ColumnName* is null-terminated.

Usage:

`SQLProcedureColumns()` returns the information in a result set, ordered by PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, and COLUMN_TYPE. "Columns returned by SQLProcedureColumns" on page 253 lists the columns in the result set. Applications should be aware that columns beyond the last column might be defined in future releases.

Since calls to `SQLProcedureColumns()` in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, and COLUMN_NAME columns supported by the connected DBMS.

If the SQL_ATTR_LONGDATA_COMPAT connection attribute is set, LOB column types will be reported as LONG VARCHAR, LONG VARBINARY or LONG VARGRAPHIC types.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

If the stored procedure is at a DB2 for MVS/ESA V4.1 up to V6 server, the name of the stored procedure must be registered in the server's SYSIBM.SYSPROCEDURES catalog table. For V7 and later servers, the stored procedures must be registered in the server's SYSIBM.SYSROUTINES and SYSIBM.SYSPARAMS catalog tables.

For versions of other DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set will be returned.

DB2 CLI will return information on the input, input/output, and output parameters associated with the stored procedure, but cannot return descriptor information for any result sets that the stored procedure might return.

Columns returned by SQLProcedureColumns

Column 1 PROCEDURE_CAT (VARCHAR(128))
Procedure catalog name. The value is NULL if this procedure does not have catalogs.

Column 2 PROCEDURE_SCHEM (VARCHAR(128))
Name of the schema containing PROCEDURE_NAME. (This is also NULL for DB2 for MVS/ESA V 4.1 or later `SQLProcedureColumns()` result sets.)

Column 3 PROCEDURE_NAME (VARCHAR(128))
Name of the procedure.

Column 4 COLUMN_NAME (VARCHAR(128))
Name of the parameter.

Column 5 COLUMN_TYPE (SMALLINT not NULL)
Identifies the type information associated with this row. The values can be:

- SQL_PARAM_TYPE_UNKNOWN : the parameter type is unknown.

  Note: This is not returned.
- SQL_PARAM_INPUT : this parameter is an input parameter.
- SQL_PARAM_INPUT_OUTPUT : this parameter is an input / output parameter.
- SQL_PARAM_OUTPUT : this parameter is an output parameter.
- SQL_RETURN_VALUE : the procedure column is the return value of the procedure.

  Note: This is not returned.
- SQL_RESULT_COL : this parameter is actually a column in the result set.

  Note: This is not returned.

Column 6 DATA_TYPE (SMALLINT not NULL)
SQL data type.

Column 7 TYPE_NAME (VARCHAR(128) not NULL)
Character string representing the name of the data type corresponding to DATA_TYPE.

Column 8 COLUMN_SIZE (INTEGER)
For XML arguments in SQL routines, zero is returned (as XML arguments have no length). For cataloged external routines, however, XML parameters are declared as XML AS CLOB(n), in which case COLUMN_SIZE is the cataloged length, n.

If the DATA_TYPE column value denotes a character or binary string, then this column contains the maximum length in SQLCHAR or SQLWCHAR elements; if it is a graphic (DBCS) string, this is the number of double byte SQLCHAR or SQLWCHAR elements for the parameter.

For date, time, timestamp data types, this is the total number of SQLCHAR or SQLWCHAR elements required to display the value when converted to character.

For numeric data types, this is either the total number of digits, or the total number of bits allowed in the column, depending on the value in the NUM_PREC_RADIX column in the result set.

See also the table of data type precision.

Column 9 BUFFER_LENGTH (INTEGER)
The maximum number of bytes for the associated C buffer to store data from this parameter if SQL_C_DEFAULT were specified on the `SQLBindCol()`, `SQLGetData()` and `SQLBindParameter()` calls. This length excludes any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

For XML arguments in SQL routines, zero is returned (as XML arguments have no length). For cataloged external routines, however, XML parameters are declared as XML AS CLOB(n), in which case BUFFER_LENGTH is the cataloged length, n.

See the table of data type length.

Column 10 DECIMAL_DIGITS (SMALLINT)
The scale of the parameter. NULL is returned for data types where scale is not applicable.

See the table of data type scale.

Column 11 NUM_PREC_RADIX (SMALLINT)
Either 10 or 2 or NULL. If DATA_TYPE is an approximate numeric data type, this column contains the value 2, then the COLUMN_SIZE column contains the number of bits allowed in the parameter.

If DATA_TYPE is an exact numeric data type, this column contains the value 10 and the COLUMN_SIZE and DECIMAL_DIGITS columns contain the number of decimal digits allowed for the parameter.

For numeric data types, the DBMS can return a NUM_PREC_RADIX of either 10 or 2.

NULL is returned for data types where radix is not applicable.

Column 12 NULLABLE (SMALLINT not NULL)
SQL_NO_NULLS if the parameter does not accept NULL values.

SQL_NULLABLE if the parameter accepts NULL values.

Column 13 REMARKS (VARCHAR(254))
Might contain descriptive information about the parameter.

Column 14 COLUMN_DEF (VARCHAR)
The default value of the column.

If NULL was specified as the default value, then this column is the word NULL, not enclosed in quotation marks. If the default value cannot be represented without truncation, then this column contains TRUNCATED, with no enclosing single quotation marks. If no default value was specified, then this column is NULL.

The value of COLUMN_DEF can be used in generating a new column definition, except when it contains the value TRUNCATED.

Column 15 SQL_DATA_TYPE (SMALLINT not NULL)
The value of the SQL data type as it appears in the SQL_DESC_TYPE field of the descriptor. This column is the same as the DATA_TYPE column except for datetime data types (DB2 CLI does not support interval data types).

For datetime data types, the SQL_DATA_TYPE field in the result set will be SQL_DATETIME, and the SQL_DATETIME_SUB field will return the subcode for the specific datetime data type (SQL_CODE_DATE, SQL_CODE_TIME or SQL_CODE_TIMESTAMP).

Column 16 SQL_DATETIME_SUB (SMALLINT)
The subtype code for datetime data types. For all other data types this column returns a NULL (including interval data types which DB2 CLI does not support).

Column 17 CHAR_OCTET_LENGTH (INTEGER)
The maximum length in bytes of a character data type column. For all other data types, this column returns a NULL.

Column 18 ORDINAL_POSITION (INTEGER NOT NULL)
Contains the ordinal position of the parameter given by COLUMN_NAME in this result set. This is the ordinal position of the argument to be provided on the CALL statement. The leftmost argument has an ordinal position of 1.

Column 19 IS_NULLABLE (Varchar)
- "NO" if the column does not include NULLs.
- "YES" if the column can include NULLs.
- zero-length string if nullability is unknown.

ISO rules are followed to determine nullability.

An ISO SQL-compliant DBMS cannot return an empty string.

The value returned for this column is different than the value returned for the NULLABLE column. (See the description of the NULLABLE column.)

Note: The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents and order are identical to those defined for the `SQLProcedureColumns()` result set in ODBC.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 118. SQLProcedureColumns SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
42601	PARMLIST syntax error.	The PARMLIST value in the stored procedures catalog table contains a syntax error.

Table 118. SQLProcedureColumns SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

`SQLProcedureColumns()` does not return information about the attributes of result sets that might be returned from stored procedures.

If an application is connected to a DB2 server that does not provide support for a stored procedure catalog, or does not provide support for stored procedures, `SQLProcedureColumns()` will return an empty result set.

Example:

```
/* get input/output parameter information for a procedure */
sqlrc = SQLProcedureColumns(hstmt,
                            NULL,
                            0, /* catalog name not used */
                            (unsigned char *)colSchemaNamePattern,
                            SQL_NTS, /* schema name not currently used */
                            (unsigned char *)procname,
                            SQL_NTS,
                            colNamePattern,
                            SQL_NTS); /* all columns */
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "Connection attributes (CLI) list" on page 326
- "Data type length (CLI) table" on page 391
- "Data type precision (CLI) table" on page 389
- "Data type scale (CLI) table" on page 390
- "SQLProcedures function (CLI) - Get list of procedure names" on page 257

Related samples:

- "spcall.c -- Call individual stored procedures"

SQLProcedures function (CLI) - Get list of procedure names

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLProcedures()` returns a list of stored procedure names that have been registered at the server, and which match the specified search pattern.

The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLProceduresW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLProcedures    (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szProcCatalog */
               SQLSMALLINT       NameLength1,       /* cbProcCatalog */
               SQLCHAR           *SchemaName,       /* szProcSchema */
               SQLSMALLINT       NameLength2,       /* cbProcSchema */
               SQLCHAR           *ProcName,         /* szProcName */
               SQLSMALLINT       NameLength3);      /* cbProcName */
```

Function arguments:

Table 119. SQLProcedures arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.

Table 119. SQLProcedures arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	*CatalogName*	Input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	Input	Buffer that can contain a *pattern value* to qualify the result set by schema name. For DB2 for MVS/ESA V 4.1 and above, all the stored procedures are in one schema; the only acceptable value for the *SchemaName* argument is a null pointer. If a value is specified, an empty result set and SQL_SUCCESS are returned. For DB2 Database for Linux, UNIX, and Windows, *SchemaName* can contain a valid pattern value. For more information about valid search patterns, refer to the catalog functions input arguments.
SQLSMALLINT	*NameLength2*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*ProcName*	Input	Buffer that can contain a *pattern value* to qualify the result set by table name.
SQLSMALLINT	*NameLength3*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *ProcName*, or SQL_NTS if *ProcName* is null-terminated.

Usage:

The result set returned by `SQLProcedures()` contains the columns listed in "Columns returned by SQLProcedures" on page 259 in the order given. The rows are ordered by PROCEDURE_CAT, PROCEDURE_SCHEMA, and PROCEDURE_NAME.

Since calls to `SQLProcedures()` in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

If the SQL_ATTR_LONGDATA_COMPAT connection attribute is set, LOB column types will be reported as LONG VARCHAR, LONG VARBINARY, or LONG VARGRAPHIC types.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

If the stored procedure is at a DB2 for MVS/ESA V4.1 up to V6 server, the name of the stored procedures must be registered in the server's SYSIBM.SYSPROCEDURES catalog table. For V7 and later servers, the stored procedure must be registered in the server's SYSIBM.SYSROUTINES and SYSIBM.SYSPARAMS catalog tables.

For other versions of DB2 servers that do not provide facilities for a stored procedure catalog, an empty result set will be returned.

Columns returned by SQLProcedures

Column 1 PROCEDURE_CAT (VARCHAR(128))
Procedure catalog name. The value is NULL if this procedure does not have catalogs.

Column 2 PROCEDURE_SCHEM (VARCHAR(128))
The name of the schema containing PROCEDURE_NAME.

Column 3 PROCEDURE_NAME (VARCHAR(128) NOT NULL)
The name of the procedure.

Column 4 NUM_INPUT_PARAMS (INTEGER not NULL)
Number of input parameters. INOUT parameters are not counted as part of this number.

To determine information regarding INOUT parameters, examine the COLUMN_TYPE column returned by `SQLProcedureColumns()`.

Column 5 NUM_OUTPUT_PARAMS (INTEGER not NULL)
Number of output parameters. INOUT parameters are not counted as part of this number.

To determine information regarding INOUT parameters, examine the COLUMN_TYPE column returned by `SQLProcedureColumns()`.

Column 6 NUM_RESULT_SETS (INTEGER not NULL)
Number of result sets returned by the procedure.

This column should not be used, it is reserved for future use by ODBC.

Column 7 REMARKS (VARCHAR(254))
Contains the descriptive information about the procedure.

Column 8 PROCEDURE_TYPE (SMALLINT)
Defines the procedure type:

- SQL_PT_UNKNOWN: It cannot be determined whether the procedure returns a value.
- SQL_PT_PROCEDURE: The returned object is a procedure; that is, it does not have a return value.
- SQL_PT_FUNCTION: The returned object is a function; that is, it has a return value.

DB2 CLI always returns SQL_PT_PROCEDURE.

Note: The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents and order are identical to those defined for the `SQLProcedures()` result set in ODBC.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 120. SQLProcedures SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

If an application is connected to a DB2 server that does not provide support for a stored procedure catalog, or does not provide support for stored procedures, `SQLProcedureColumns()` will return an empty result set.

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Connection attributes (CLI) list" on page 326
- "SQLProcedureColumns function (CLI) - Get input/output parameter information for a procedure" on page 251
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

SQLPutData function (CLI) - Passing data value for a parameter

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	ISO CLI

`SQLPutData()` is called following an `SQLParamData()` call returning SQL_NEED_DATA to supply parameter data values. This function can be used to send large parameter values in pieces.

Syntax:

```
SQLRETURN   SQLPutData       (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLPOINTER        DataPtr,           /* rgbValue */
               SQLLEN            StrLen_or_Ind);    /* cbValue */
```

Function arguments:

Table 121. SQLPutData arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLPOINTER	*DataPtr*	Input	Pointer to the actual data, or portion of data, for a parameter. The data must be in the form specified in the `SQLBindParameter()` call that the application used when specifying the parameter.

Table 121. SQLPutData arguments (continued)

Data type	Argument	Use	Description
SQLLEN	*StrLen_or_Ind*	Input	The length of *DataPtr*. Specifies the amount of data sent in a call to `SQLPutData()` . The amount of data can vary with each call for a given parameter. The application can also specify SQL_NTS or SQL_NULL_DATA for *StrLen_or_Ind*. *StrLen_or_Ind* is ignored for all fixed length C buffer types, such as date, time, timestamp, and all numeric C buffer types. For cases where the C buffer type is SQL_C_CHAR or SQL_C_BINARY, or if SQL_C_DEFAULT is specified as the C buffer type and the C buffer type default is SQL_C_CHAR or SQL_C_BINARY, this is the number of bytes of data in the *DataPtr* buffer.

Usage:

The application calls `SQLPutData()` after calling `SQLParamData()` on a statement in the SQL_NEED_DATA state to supply the data values for an SQL_DATA_AT_EXEC parameter. Long data can be sent in pieces via repeated calls to `SQLPutData()`. DB2 CLI generates a temporary file for each SQL_DATA_AT_EXEC parameter to which each piece of data is appended when `SQLPutData()` is called. The path in which DB2 CLI creates its temporary files can be set using the TEMPDIR keyword in the `db2cli.ini` file. If this keyword is not set, DB2 CLI attempts to write to the path specified by the environment variables TEMP or TMP. After all the pieces of data for the parameter have been sent, the application calls `SQLParamData()` again to proceed to the next SQL_DATA_AT_EXEC parameter, or, if all parameters have data values, to execute the statement.

`SQLPutData()` cannot be called more than once for a fixed length C buffer type, such as SQL_C_LONG.

After an `SQLPutData()` call, the only legal function calls are `SQLParamData()`, `SQLCancel()`, or another `SQLPutData()` if the input data is character or binary data. As with `SQLParamData()`, all other function calls using this statement handle will fail. In addition, all function calls referencing the parent connection handle of *StatementHandle* will fail if they involve changing any attribute or state of that connection; that is, the following function calls on the parent connection handle are also not permitted:

- SQLSetConnectAttr()
- SQLEndTran()

Should they be invoked during an SQL_NEED_DATA sequence, these functions will return SQL_ERROR with SQLSTATE of **HY**010 and the processing of the SQL_DATA_AT_EXEC parameters will not be affected.

If one or more calls to `SQLPutData()` for a single parameter results in SQL_SUCCESS, attempting to call `SQLPutData()` with *StrLen_or_Ind* set to SQL_NULL_DATA for the same parameter results in an error with SQLSTATE of **22**005. This error does not result in a change of state; the statement handle is still in a *Need Data* state and the application can continue sending parameter data.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Some of the following diagnostics conditions might also be reported on the final `SQLParamData()` call rather than at the time the `SQLPutData()` is called.

Table 122. SQLPutData SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated.	The data sent for a numeric parameter was truncated without the loss of significant digits. Timestamp data sent for a date or time column was truncated. Function returns with SQL_SUCCESS_WITH_INFO.
22001	String data right truncation.	More data was sent for a binary or char data than the data source can support for that column.
22003	Numeric value out of range.	The data sent for a numeric parameter caused the whole part of the number to be truncated when assigned to the associated column. `SQLPutData()` was called more than once for a fixed length parameter.
22005	Error in assignment.	The data sent for a parameter was incompatible with the data type of the associated table column.
22007	Invalid datetime format.	The data value sent for a date, time, or timestamp parameters was invalid.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	The argument *DataPtr* was a NULL pointer, and the argument *StrLen_or_Ind* was neither 0 nor SQL_NULL_DATA.
HY010	Function sequence error.	The statement handle *StatementHandle* must be in a need data state and must have been positioned on an SQL_DATA_AT_EXEC parameter via a previous `SQLParamData()` call.
HY090	Invalid string or buffer length.	The argument *DataPtr* was not a NULL pointer, and the argument *StrLen_or_Ind* was less than 0, but not equal to SQL_NTS or SQL_NULL_DATA.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

A additional value for *StrLen_or_Ind*, SQL_DEFAULT_PARAM, was introduced in ODBC 2.0, to indicate that the procedure is to use the default value of a parameter, rather than a value sent from the application. Since DB2 stored procedure arguments do not support default values, specification of this value for StrLen_or_Ind argument will result in an error when the CALL statement is executed since the SQL_DEFAULT_PARAM value will be considered an invalid length.

ODBC 2.0 also introduced the SQL_LEN_DATA_AT_EXEC(*length*) macro to be used with the *StrLen_or_Ind argument*. The macro is used to specify the sum total length of the entire data that would be sent for character or binary C data via the subsequent `SQLPutData()` calls. Since the DB2 ODBC driver does not need this information, the macro is not needed. An ODBC application calls `SQLGetInfo()` with the SQL_NEED_LONG_DATA_LEN option to check if the driver needs this information. The DB2 ODBC driver will return 'N' to indicate that this information is not needed by `SQLPutData()`.

Example:

```
SQLCHAR buffer[BUFSIZ];
size_t n = BUFSIZ;

/* ... */

/* passing data value for a parameter */
cliRC = SQLPutData(hstmt, buffer, n);
```

Related concepts:

- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Long data for bulk inserts and updates in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLCancel function (CLI) - Cancel statement" on page 49
- "SQLNativeSql function (CLI) - Get native SQL text" on page 231
- "SQLParamData function (CLI) - Get next parameter for which a data value is needed" on page 239
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dtlob.c -- How to read and write LOB data"

SQLRowCount function (CLI) - Get row count

Purpose:

Specification:	**DB2 CLI** 1.1	**ODBC** 1.0	**ISO CLI**

SQLRowCount() returns the number of rows in a table that were affected by an UPDATE, INSERT, DELETE, or MERGE statement executed against the table, or a view based on the table.

SQLExecute() or SQLExecDirect() must be called before calling this function.

Syntax:

```
SQLRETURN   SQLRowCount (
               SQLHSTMT     StatementHandle,   /* hstmt */
               SQLLEN       *RowCountPtr);     /* pcrow */
```

Function arguments:

Table 123. SQLRowCount arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLLEN *	*RowCountPtr*	output	Pointer to location where the number of rows affected is stored.

Usage:

If the last executed statement referenced by the input statement handle was not an UPDATE, INSERT, DELETE, or MERGE statement, or if it did not execute successfully, then the function sets the contents of *RowCountPtr* to -1.

Any rows in other tables that might have been affected by the statement (for example, cascading deletes) are not included in the count.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 124. SQLRowCount SQLSTATEs

SQLSTATE	Description	Explanation
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The function was called prior to calling SQLExecute() or SQLExecDirect() for the *StatementHandle*.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.

Authorization:

None.

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLExecute function (CLI) - Execute a statement" on page 106
- "DELETE statement" in *SQL Reference, Volume 2*
- "INSERT scalar function" in *SQL Reference, Volume 1*
- "UPDATE statement" in *SQL Reference, Volume 2*

SQLSetColAttributes function (CLI) - Set column attributes

Deprecated:

Note:

In ODBC 3.0, `SQLSetColAttributes()` has been deprecated, and DB2 CLI no longer supports this function.

Now that DB2 CLI uses deferred prepare by default, there is no need for the functionality of `SQLSetColAttributes()`.

Related concepts:

- "Deferred prepare in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI and ODBC function summary" on page 1

SQLSetConnectAttr function (CLI) - Set connection attributes

Purpose:

Specification:	**DB2 CLI** 5.0	**ODBC** 3.0	**ISO CLI**

`SQLSetConnectAttr()` sets attributes that govern aspects of connections.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLSetConnectAttrW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLSetConnectAttr (
               SQLHDBC          ConnectionHandle,  /* hdbc */
               SQLINTEGER       Attribute,         /* fOption */
               SQLPOINTER       ValuePtr,          /* pvParam */
               SQLINTEGER       StringLength);     /* fStrLen */
```

Function arguments:

Table 125. SQLSetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	Connection handle.
SQLINTEGER	*Attribute*	input	Attribute to set, listed in the connection attributes list.
SQLPOINTER	*ValuePtr*	input	Pointer to the value to be associated with *Attribute*. Depending on the value of *Attribute*, *ValuePtr* will be a 32-bit unsigned integer value or pointer to a null-terminated character string. Note that if the *Attribute* argument is a driver-specific value, the value in **ValuePtr* can be a signed integer. Refer to the connection attributes list for details.
SQLINTEGER	*StringLength*	input	If *Attribute* is an ODBC-defined attribute and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. For character string data, *StringLength* should contain the number of bytes in the string. If *Attribute* is an ODBC-defined attribute and *ValuePtr* is an integer, *StringLength* is ignored. If *Attribute* is a DB2 CLI attribute, the application indicates the nature of the attribute by setting the *StringLength* argument. *StringLength* can have the following values: • If *ValuePtr* is a pointer to a character string, then *StringLength* is the number of bytes needed to store the string or SQL_NTS. • If *ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *StringLength*. This places a negative value in *StringLength*. • If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *StringLength* should have the value SQL_IS_POINTER. • If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate.

Usage:

Setting statement attributes using SQLSetConnectAttr() no longer supported

The ability to set statement attributes using `SQLSetConnectAttr()` is no longer supported. To support applications written before version 5, some statement attributes can be set using `SQLSetConnectAttr()` in this release of DB2 CLI. All applications that rely on this behavior, however, should be updated to use `SQLSetStmtAttr()` instead.

If `SQLSetConnectAttr()` is called to set a statement attribute that sets the header field of a descriptor, the descriptor field is set for the application descriptors currently associated with all statements on the connection. However, the attribute setting does not affect any descriptors that might be associated with the statements on that connection in the future.

Connection Attributes

At any time between allocating and freeing a connection, an application can call `SQLSetConnectAttr()`. All connection and statement attributes successfully set by the application for the connection persist until `SQLFreeHandle()` is called on the connection.

Some connection attributes can be set only before a connection has been made; others can be set only after a connection has been made, while some cannot be set once a statement is allocated. Refer to the connection attributes list for details on when each attribute can be set.

Some connection attributes support substitution of a similar value if the data source does not support the value specified in *ValuePtr*. In such cases, DB2 CLI returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed.). To determine the substituted value, an application calls `SQLGetConnectAttr()`.

The format of information set through *ValuePtr* depends on the specified *Attribute*. `SQLSetConnectAttr()` will accept attribute information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of each is noted in the attribute's description. Character strings pointed to by the *ValuePtr* argument of `SQLSetConnectAttr()` have a length of *StringLength* bytes. The *StringLength* argument is ignored if the length is defined by the attribute.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

DB2 CLI can return SQL_SUCCESS_WITH_INFO to provide information about the result of setting an option.

When *Attribute* is a statement attribute, `SQLSetConnectAttr()` can return any SQLSTATEs returned by `SQLSetStmtAttr()`.

Table 126. SQLSetConnectAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	General error.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed.	DB2 CLI did not support the value specified in **ValuePtr* and substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08002	Connection in use.	The argument *Attribute* was SQL_ATTR_ODBC_CURSORS and DB2 CLI was already connected to the data source.
08003	Connection is closed.	An *Attribute* value was specified that required an open connection, but the *ConnectionHandle* was not in a connected state.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.

Table 126. SQLSetConnectAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	The argument *Attribute* was SQL_ATTR_CURRENT_QUALIFIER and a result set was pending.
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by `SQLGetDiagRec()` in the *MessageText buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	A null pointer was passed for *ValuePtr* and the value in **ValuePtr* was a string value.
HY010	Function sequence error.	An asynchronously executing function was called for a *StatementHandle* associated with the *ConnectionHandle* and was still executing when `SQLSetConnectAttr()` was called. `SQLExecute()` or `SQLExecDirect()` was called for a *StatementHandle* associated with the *ConnectionHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. `SQLBrowseConnect()` was called for the *ConnectionHandle* and returned SQL_NEED_DATA. This function was called before `SQLBrowseConnect()` returned SQL_SUCCESS_WITH_INFO or SQL_SUCCESS.
HY011	Operation invalid at this time.	The argument *Attribute* was SQL_ATTR_TXN_ISOLATION and a transaction was open.
HY024	Invalid attribute value.	Given the specified *Attribute* value, an invalid value was specified in **ValuePtr*. (DB2 CLI returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE. For all other connection and statement attributes, DB2 CLI must verify the value specified in *ValuePtr*.) The *Attribute* argument was SQL_ATTR_TRACEFILE or SQL_ATTR_TRANSLATE_LIB, and **ValuePtr* was an empty string.
HY090	Invalid string or buffer length.	The *StringLength* argument was less than 0, but was not SQL_NTS.
HY092	Option type out of range.	The value specified for the argument *Attribute* was not valid for this version of DB2 CLI.
HYC00	Driver not capable.	The value specified for the argument *Attribute* was a valid connection or statement attribute for the version of the DB2 CLI driver, but was not supported by the data source.

Restrictions:

None.

Example:

```
/* set AUTOCOMMIT on */
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_AUTOCOMMIT,
```

```
                          (SQLPOINTER)SQL_AUTOCOMMIT_ON,
                          SQL_NTS);
/* ... */

/* set AUTOCOMMIT OFF */
cliRC = SQLSetConnectAttr(hdbc,
                          SQL_ATTR_AUTOCOMMIT,
                          (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
                          SQL_NTS);
```

Related concepts:

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "Connection attributes (CLI) list" on page 326
- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLGetConnectAttr function (CLI) - Get current attribute setting" on page 146
- "SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute" on page 216
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294

Related samples:

- "tbread.c -- How to read data from tables"
- "dbuse.c -- How to use a database"

SQLSetConnection function (CLI) - Set connection handle

Purpose:

Specification:	**DB2 CLI** 2.1		

This function is needed if the application needs to deterministically switch to a particular connection before continuing execution. It should only be used when the application is mixing DB2 CLI function calls with embedded SQL function calls and where multiple connections are used.

Syntax:

```
SQLRETURN   SQLSetConnection (SQLHDBC          ConnectionHandle); /* hdbc */
```

Function arguments:

Table 127. SQLSetConnection arguments

Data type	Argument	Use	Description
SQLHDBC	*ConnectionHandle*	input	The connection handle associated with the connection that the application wishes to switch to.

Usage:

In DB2 CLI version 1 it was possible to mix DB2 CLI calls with calls to routines containing embedded SQL as long as the connect request was issued via the DB2 CLI connect function. The embedded SQL routine would simply use the existing DB2 CLI connection.

Although this is still true, there is a potential complication: DB2 CLI allows multiple concurrent connections. This means that it is no longer clear which connection an embedded SQL routine would use upon being invoked. In practice, the embedded routine would use the connection associated with the most recent network activity. However, from the application's perspective, this is not always deterministic and it is difficult to keep track of this information. `SQLSetConnection()` is used to allow the application to *explicitly* specify which connection is active. The application can then call the embedded SQL routine.

`SQLSetConnection()` is not needed if the application makes use of DB2 CLI exclusively. Under those conditions, each statement handle is implicitly associated with a connection handle and there is never any confusion as to which connection a particular DB2 CLI function applies.

Return codes:
- SQL_SUCCESS
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 128. SQLSetConnection SQLSTATEs

SQLSTATE	Description	Explanation
08003	Connection is closed.	The connection handle provided is not currently associated with an open connection to a database server.
HY000	General error.	An error occurred for which there was no specific SQLSTATE and for which no implementation-specific SQLSTATE was defined. The error message returned by `SQLGetDiagRec()` in the argument *MessageText* describes the error and its cause.

Restrictions:

None.

Example:

```
/* perform statements on the first connection */
cliRC = SQLSetConnection(hdbc1);

/* ... */

/* perform statements on the second connection */
cliRC = SQLSetConnection(hdbc2);
```

Related concepts:
- "Considerations for mixing embedded SQL and DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Multisite updates (two phase commit) in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLConnect function (CLI) - Connect to a data source" on page 73
- "SQLDriverConnect function (CLI) - (Expanded) Connect to a data source" on page 91
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbmconx.c -- How to use multiple databases with embedded SQL."

SQLSetConnectOption function (CLI) - Set connection option

Deprecated:

Note:

In ODBC 3.0, `SQLSetConnectOption()` has been deprecated and replaced with `SQLSetConnectAttr()`.

Although this version of DB2 CLI continues to support `SQLSetConnectOption()`, we recommend that you use `SQLSetConnectAttr()` in your DB2 CLI programs so that they conform to the latest standards.

This deprecated function cannot be used in a 64-bit environment.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLSetConnectOptionW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Migrating to the new function

The statement:

```
SQLSetConnectOption(
           hdbc,
           SQL_AUTOCOMMIT,
           SQL_AUTOCOMMIT_OFF);
```

for example, would be rewritten using the new function as:

```
SQLSetConnectAttr(
           hdbc,
           SQL_ATTR_AUTOCOMMIT,
           SQL_AUTOCOMMIT_OFF,
           0);
```

Related concepts:

- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266

SQLSetCursorName function (CLI) - Set cursor name

Purpose:

Specification:	DB2 CLI 1.1	ODBC 1.0	ISO CLI

SQLSetCursorName() associates a cursor name with the statement handle. This function is optional because DB2 CLI implicitly generates a cursor name. The implicit cursor name is available after the dynamic SQL has been prepared on the statement handle.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSetCursorNameW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLSetCursorName (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLCHAR           *CursorName,      /* szCursor */
               SQLSMALLINT       NameLength);      /* cbCursor */
```

Function arguments:

Table 129. SQLSetCursorName arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle
SQLCHAR *	*CursorName*	input	Cursor name
SQLSMALLINT	*NameLength*	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store the *CursorName* argument.

Usage:

DB2 CLI always generates and uses an internally generated cursor name when a query is prepared or executed directly. SQLSetCursorName() allows an application-defined cursor name to be used in an SQL statement (a positioned UPDATE or DELETE). DB2 CLI maps this name to the internal name. The name will remain associated with the statement handle, until the handle is dropped, or another SQLSetCursorName() is called on this statement handle.

Although SQLGetCursorName() will return the name set by the application (if one was set), error messages associated with positioned UPDATE and DELETE statements will refer to the internal name. For this reason, we recommend that you do not use SQLSetCursorName() for positioned UPDATEs and DELETEs, but instead use the internal name which can be obtained by calling SQLGetCursorName().

Cursor names must follow these rules:

- All cursor names within the connection must be unique.
- Each cursor name must be less than or equal to 18 bytes in length. Any attempt to set a cursor name longer than 18 bytes results in truncation of that cursor name to 18 bytes. (No warning is generated.)
- Since internally generated names begin with SQLCUR or SQL_CUR, the application must not input a cursor name starting with either SQLCUR or SQL_CUR in order to avoid conflicts with internal names.

- Since a cursor name is considered an identifier in SQL, it must begin with an English letter (a-z, A-Z) followed by any combination of digits (0-9), English letters or the underscore character (_).
- To permit cursor names containing characters other than those listed above (such as National Language Set or Double Bytes Character Set characters), the application must enclose the cursor name in double quotes (").
- Unless the input cursor name is enclosed in double quotes, all leading and trailing blanks from the input cursor name string will be removed.

For efficient processing, applications should not include any leading or trailing spaces in the *CursorName* buffer. If the *CursorName* buffer contains a delimited identifier, applications should position the first double quote as the first character in the *CursorName* buffer.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 130. SQLSetCursorName SQLSTATEs

SQLSTATE	Description	Explanation
34000	Invalid cursor name.	The cursor name specified by the argument *CursorName* was invalid. The cursor name either begins with "SQLCUR" or "SQL_CUR" or violates the cursor naming rules (Must begin with a-z or A-Z followed by any combination of English letters, digits, or the '_' character. The cursor name specified by the argument *CursorName* already exists. The cursor name length is greater than the value returned by `SQLGetInfo()` with the SQL_MAX_CURSOR_NAME_LEN argument.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
58004	Unexpected system failure.	Unrecoverable system error.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	*CursorName* was a null pointer.

Table 130. SQLSetCursorName SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	There is an open or positioned cursor on the statement handle. The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY090	Invalid string or buffer length.	The argument *NameLength* was less than **0**, but not equal to SQL_NTS.

Authorization:

None.

Example:

```
/* set the name of the cursor */
rc = SQLSetCursorName(hstmtSelect, (SQLCHAR *)"CURSNAME", SQL_NTS);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Handles in CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "National language support and application development considerations" in *Developing SQL and External Routines*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Preparing and executing SQL statements in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Updating and deleting data in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLGetCursorName function (CLI) - Get cursor name" on page 149
- "DBCS character sets" in *Developing SQL and External Routines*

Related samples:

- "tbmod.c -- How to modify table data"

SQLSetDescField function (CLI) - Set a single field of a descriptor record

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLSetDescField()` sets the value of a single field of a descriptor record.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLSetDescFieldW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLSetDescField  (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       FieldIdentifier,
                              SQLPOINTER        ValuePtr,
                              SQLINTEGER        BufferLength);
```

Function arguments:

Table 131. SQLSetDescField arguments

Data type	Argument	Use	Description
SQLHDESC	*DescriptorHandle*	input	Descriptor handle.
SQLSMALLINT	*RecNumber*	input	Indicates the descriptor record containing the field that the application seeks to set. Descriptor records are numbered from 0, with record number 0 being the bookmark record. The *RecNumber* argument is ignored for header fields.
SQLSMALLINT	*FieldIdentifier*	input	Indicates the field of the descriptor whose value is to be set. For more information, refer to the list of values for the descriptor *FieldIdentifier* argument.
SQLPOINTER	*ValuePtr*	input	Pointer to a buffer containing the descriptor information, or a four-byte value. The data type depends on the value of *FieldIdentifier*. If *ValuePtr* is a four-byte value, either all four of the bytes are used, or just two of the four are used, depending on the value of the *FieldIdentifier* argument.

Table 131. SQLSetDescField arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	*BufferLength*	input	If *FieldIdentifier* is an ODBC-defined field and *ValuePtr* points to a character string or a binary buffer, this argument should be the length of **ValuePtr*. For character string data, *BufferLength* should contain the number of bytes in the string. If *FieldIdentifier* is an ODBC-defined field and *ValuePtr* is an integer, *BufferLength* is ignored. If *FieldIdentifier* is a driver-defined field, the application indicates the nature of the field by setting the *BufferLength* argument. *BufferLength* can have the following values: • If *ValuePtr* is a pointer to a character string, then *BufferLength* is the number of bytes needed to store the string or SQL_NTS. • If *ValuePtr* is a pointer to a binary buffer, then the application places the result of the SQL_LEN_BINARY_ATTR(length) macro in *BufferLength* This places a negative value in *BufferLength*. • If *ValuePtr* is a pointer to a value other than a character string or a binary string, then *BufferLength* should have the value SQL_IS_POINTER. • If *ValuePtr* contains a fixed-length value, then *BufferLength* is either SQL_IS_INTEGER, SQL_IS_UINTEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT, as appropriate.

Usage:

An application can call `SQLSetDescField()` to set any descriptor field one at a time. One call to `SQLSetDescField()` sets a single field in a single descriptor. This function can be called to set any field in any descriptor type, provided the field can be set. See the descriptor header and record field initialization values for more information.

Note: If a call to `SQLSetDescField()` fails, the contents of the descriptor record identified by the *RecNumber* argument are undefined.

Other functions can be called to set multiple descriptor fields with a single call of the function. The `SQLSetDescRec()` function sets a variety of fields that affect the data type and buffer bound to a column or parameter (the SQL_DESC_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE, SQL_DESC_OCTET_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE, SQL_DESC_DATA_PTR, SQL_DESC_OCTET_LENGTH_PTR, and SQL_DESC_INDICATOR_PTR fields). `SQLBindCol()` or `SQLBindParameter()` can be used to make a complete specification for the binding of a column or parameter. These functions each set a specific group of descriptor fields with one function call.

`SQLSetDescField()` can be called to change the binding buffers by adding an offset to the binding pointers (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, or SQL_DESC_OCTET_LENGTH_PTR). This changes the binding buffers without

calling SQLBindCol() or SQLBindParameter(). This allows an application to quickly change SQL_DESC_DATA_PTR without concern for changing other fields, for instance SQL_DESC_DATA_TYPE.

Descriptor header fields are set by calling SQLSetDescField() with a *RecNumber* of 0, and the appropriate *FieldIdentifier*. Many header fields contain statement attributes, so can also be set by a call to SQLSetStmtAttr(). This allows applications to set a statement attribute without first obtaining a descriptor handle. A *RecNumber* of 0 is also used to set bookmark fields.

Note: The statement attribute SQL_ATTR_USE_BOOKMARKS should always be set before calling SQLSetDescField() to set bookmark fields. While this is not mandatory, it is strongly recommended.

Sequence of setting descriptor fields

When setting descriptor fields by calling SQLSetDescField(), the application must follow a specific sequence:

- The application must first set the SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE, or SQL_DESC_DATETIME_INTERVAL_CODE field.

  Note: SQL_DESC_DATETIME_INTERVAL_CODE is defined by ODBC but not supported by DB2 CLI.
- After one of these fields has been set, the application can set an attribute of a data type, and the driver sets data type attribute fields to the appropriate default values for the data type. Automatic defaulting of type attribute fields ensures that the descriptor is always ready to use once the application has specified a data type. If the application explicitly sets a data type attribute, it is overriding the default attribute.
- After one of the fields listed in Step 1 has been set, and data type attributes have been set, the application can set SQL_DESC_DATA_PTR. This prompts a consistency check of descriptor fields. If the application changes the data type or attributes after setting the SQL_DESC_DATA_PTR field, then the driver sets SQL_DESC_DATA_PTR to a null pointer, unbinding the record. This forces the application to complete the proper steps in sequence, before the descriptor record is usable.

Initialization of descriptor fields

When a descriptor is allocated, the fields in the descriptor can be initialized to a default value, be initialized without a default value, or be undefined for the type of descriptor. Refer to the list of descriptor header and record field initialization values for details.

The fields of an IRD have a default value only after the statement has been prepared or executed and the IRD has been populated, not when the statement handle or descriptor has been allocated. Until the IRD has been populated, any attempt to gain access to a field of an IRD will return an error.

Some descriptor fields are defined for one or more, but not all, of the descriptor types (ARDs and IRDs, and APDs and IPDs). When a field is undefined for a type of descriptor, it is not needed by any of the functions that use that descriptor. Because a descriptor is a logical view of data, rather than an actual data structure, these extra fields have no effect on the defined fields.

The fields that can be accessed by SQLGetDescField() are not necessarily set by SQLSetDescField(). Fields that can be set by SQLSetDescField() are described in the descriptor header and record field initialization values list.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 132. SQLSetDescField SQLSTATEs

SQLSTATE	Description	Explanation
01000	General warning	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed.	DB2 CLI did not support the value specified in **ValuePtr* (if *ValuePtr* was a pointer) or the value in *ValuePtr* (if *ValuePtr* was a four-byte value), or **ValuePtr* was invalid because of SQL constraints or requirements, so DB2 CLI substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index.	The *FieldIdentifier* argument was a header field, and the *RecNumber* argument was not 0. The *RecNumber* argument was 0 and the *DescriptorHandle* was an IPD. The *RecNumber* argument was less than 0.
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by SQLGetDiagRec() in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The *DescriptorHandle* was associated with a StatementHandle for which an asynchronously executing function (not this one) was called and was still executing when this function was called. SQLExecute() or SQLExecDirect() was called for the *StatementHandle* with which the *DescriptorHandle* was associated and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY016	Cannot modify an implementation row descriptor.	The *DescriptorHandle* argument was associated with an IRD, and the *FieldIdentifier* argument was not SQL_DESC_ARRAY_STATUS_PTR.
HY021	Inconsistent descriptor information.	The TYPE field, or any other field associated with the TYPE field in the descriptor, was not valid or consistent. The TYPE field was not a valid DB2 CLI C type. Descriptor information checked during a consistency check was not consistent.

Table 132. SQLSetDescField SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY091	Invalid descriptor field identifier.	The value specified for the *FieldIdentifier* argument was not a DB2 CLI defined field and was not a defined value. The value specified for the *RecNumber* argument was greater than the value in the SQL_DESC_COUNT field. The *FieldIdentifier* argument was SQL_DESC_ALLOC_TYPE.
HY092	Option type out of range.	The value specified for the *Attribute* argument was not valid.
HY094	Invalid scale value.	The value specified for *pfParamType* was either SQL_DECIMAL or SQL_NUMERIC and the value specified for *DecimalDigits* was less than 0 or greater than the value for the argument *pcbColDef* (precision). The value specified for *pfParamType* was SQL_C_TIMESTAMP and the value for *pfParamType* was either SQL_CHAR or SQL_VARCHAR and the value for *DecimalDigits* was less than 0 or greater than 6.
HY105	Invalid parameter type.	The value specified for the SQL_DESC_PARAMETER_TYPE field was invalid. (For more information, see the *InputOutputType* Argument section in `SQLBindParameter()`.)

Restrictions:

None.

Example:

```
/* set a single field of a descriptor record */
rc = SQLSetDescField(hARD,
                     1,
                     SQL_DESC_TYPE,
                     (SQLPOINTER)SQL_SMALLINT,
                     SQL_IS_SMALLINT);
```

Related concepts:

- "Consistency checks for descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Descriptor FieldIdentifier argument values (CLI)" on page 367
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptor header and record field initialization values (CLI)" on page 378
- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" on page 160

- "SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record" on page 164
- "SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data" on page 281

Related samples:

- "dbuse.c -- How to use a database"

SQLSetDescRec function (CLI) - Set multiple descriptor fields for a column or parameter data

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

The SQLSetDescRec() function sets multiple descriptor fields that affect the data type and buffer bound to a column or parameter data.

Syntax:

```
SQLRETURN   SQLSetDescRec    (SQLHDESC          DescriptorHandle,
                              SQLSMALLINT       RecNumber,
                              SQLSMALLINT       Type,
                              SQLSMALLINT       SubType,
                              SQLLEN            Length,
                              SQLSMALLINT       Precision,
                              SQLSMALLINT       Scale,
                              SQLPOINTER        DataPtr,
                              SQLLEN            *StringLengthPtr,
                              SQLLEN            *IndicatorPtr);
```

Function arguments:

Table 133. SQLSetDescRec arguments

Data type	Argument	Use	Description
SQLHDESC	*DescriptorHandle*	input	Descriptor handle. This must not be an IRD handle.
SQLSMALLINT	*RecNumber*	input	Indicates the descriptor record that contains the fields to be set. Descriptor records are numbered from 0, with record number 0 being the bookmark record. This argument must be equal to or greater than 0. If *RecNumber* is greater than the value of SQL_DESC_COUNT, SQL_DESC_COUNT is changed to the value of *RecNumber*.
SQLSMALLINT	*Type*	input	The value to which to set the SQL_DESC_TYPE field for the descriptor record.
SQLSMALLINT	*SubType*	input	For records whose type is SQL_DATETIME, this is the value to which to set the SQL_DESC_DATETIME_INTERVAL_CODE field.
SQLLEN	*Length*	input	The value to which to set the SQL_DESC_OCTET_LENGTH field for the descriptor record.
SQLSMALLINT	*Precision*	input	The value to which to set the SQL_DESC_PRECISION field for the descriptor record.

Table 133. SQLSetDescRec arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*Scale*	input	The value to which to set the SQL_DESC_SCALE field for the descriptor record.
SQLPOINTER	*DataPtr*	Deferred Input or Output	The value to which to set the SQL_DESC_DATA_PTR field for the descriptor record. *DataPtr* can be set to a null pointer to set the SQL_DESC_DATA_PTR field to a null pointer.
SQLLEN *	*StringLengthPtr*	Deferred Input or Output	The value to which to set the SQL_DESC_OCTET_LENGTH_PTR field for the descriptor record. *StringLengthPtr* can be set to a null pointer to set the SQL_DESC_OCTET_LENGTH_PTR field to a null pointer.
SQLLEN *	*IndicatorPtr*	Deferred Input or Output	The value to which to set the SQL_DESC_INDICATOR_PTR field for the descriptor record. *IndicatorPtr* can be set to a null pointer to set the SQL_DESC_INDICATOR_PTR field to a null pointer.

Usage:

An application can call `SQLSetDescRec()` to set the following fields for a single column or parameter:
- SQL_DESC_TYPE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_DATA_PTR
- SQL_DESC_OCTET_LENGTH_PTR
- SQL_DESC_INDICATOR_PTR

SQL_DESC_DATETIME_INTERVAL_CODE can only be updated if SQL_DESC_TYPE indicates SQL_DATETIME.

Note: If a call to `SQLSetDescRec()` fails, the contents of the descriptor record identified by the *RecNumber* argument are undefined.

When binding a column or parameter, `SQLSetDescRec()` allows you to change multiple fields affecting the binding without calling `SQLBindCol()` or `SQLBindParameter()`, or making multiple calls to `SQLSetDescField()`. `SQLSetDescRec()` can set fields on a descriptor not currently associated with a statement. Note that `SQLBindParameter()` sets more fields than `SQLSetDescRec()`, can set fields on both an APD and an IPD in one call, and does not require a descriptor handle.

The statement attribute SQL_ATTR_USE_BOOKMARKS should always be set before calling `SQLSetDescRec()` with a *RecNumber* argument of 0 to set bookmark fields. While this is not mandatory, it is strongly recommended.

Return Codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 134. SQLSetDescRec SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index.	The *RecNumber* argument was set to 0, and the *DescriptorHandle* was an IPD handle. The *RecNumber* argument was less than 0. The *RecNumber* argument was greater than the maximum number of columns or parameters that the data source can support, and the *DescriptorHandle* argument was an APD, IPD, or ARD. The *RecNumber* argument was equal to 0, and the *DescriptorHandle* argument referred to an implicitly allocated APD. (This error does not occur with an explicitly allocated application descriptor, because it is not known whether an explicitly allocated application descriptor is an APD or ARD until execute time.)
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY010	Function sequence error.	The *DescriptorHandle* was associated with a *StatementHandle* for which an asynchronously executing function (not this one) was called and was still executing when this function was called. `SQLExecute()` or `SQLExecDirect()` was called for the *StatementHandle* with which the *DescriptorHandle* was associated and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters.
HY013	Unexpected memory handling error.	DB2 CLI was unable to access memory required to support execution or completion of the function.
HY016	Cannot modify an implementation row descriptor.	The *DescriptorHandle* argument was associated with an IRD.
HY021	Inconsistent descriptor information.	The *Type* field, or any other field associated with the TYPE field in the descriptor, was not valid or consistent. Descriptor information checked during a consistency check was not consistent.
HY094	Invalid scale value.	The value specified for *pfParamType* was either SQL_DECIMAL or SQL_NUMERIC and the value specified for *DecimalDigits* was less than 0 or greater than the value for the argument *pcbColDef* (precision). The value specified for *pfParamType* was SQL_C_TIMESTAMP and the value for *pfParamType* was either SQL_CHAR or SQL_VARCHAR and the value for *DecimalDigits* was less than 0 or greater than 6.

Restrictions:

None.

Example:

```
SQLSMALLINT type;
SQLINTEGER length, datalen;
SQLSMALLINT id_no;
/* ... */

/* set multiple descriptor fields for a column or parameter data */
rc = SQLSetDescRec(hARD, 1, type, 0, length, 0, 0, &id_no, &datalen, NULL);
```

Related concepts:

- "Consistency checks for descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLBindCol function (CLI) - Bind a column to an application variable or LOB locator" on page 10
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" on page 160
- "SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record" on page 164
- "SQLSetDescField function (CLI) - Set a single field of a descriptor record" on page 276
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "dbuse.c -- How to use a database"

SQLSetEnvAttr function (CLI) - Set environment attribute

Purpose:

Specification:	DB2 CLI 2.1		ISO CLI

`SQLSetEnvAttr()` sets an environment attribute for the current environment.

Syntax:

```
SQLRETURN   SQLSetEnvAttr    (SQLHENV          EnvironmentHandle, /* henv */
                              SQLINTEGER       Attribute,
                              SQLPOINTER       ValuePtr,          /* Value */
                              SQLINTEGER       StringLength);
```

Function arguments:

Table 135. SQLSetEnvAttr arguments

Data type	Argument	Use	Description
SQLHENV	*EnvironmentHandle*	Input	Environment handle.
SQLINTEGER	*Attribute*	Input	Environment attribute to set; refer to the list of CLI environment attributes for descriptions.
SQLPOINTER	*ValuePtr*	Input	The desired value for *Attribute*.
SQLINTEGER	*StringLength*	Input	Length of *ValuePtr* in bytes if the attribute value is a character string; if *Attribute* does not denote a string, then DB2 CLI ignores *StringLength*.

Usage:

Once set, the attribute's value affects all connections under this environment.

The application can obtain the current attribute value by calling `SQLGetEnvAttr()`.

Refer to the list of CLI environment attributes for the attributes that can be set with `SQLSetEnvAttr()`.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 136. SQLSetEnvAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY011	Operation invalid at this time.	Applications cannot set environment attributes while connection handles are allocated on the environment handle.
HY024	Invalid attribute value	Given the specified *Attribute* value, an invalid value was specified in **ValuePtr*.
HY090	Invalid string or buffer length	The *StringLength* argument was less than 0, but was not SQL_NTS.
HY092	Option type out of range.	An invalid *Attribute* value was specified.
HYC00	Driver not capable.	The specified *Attribute* is not supported by DB2 CLI. Given specified *Attribute* value, the value specified for the argument *ValuePtr* is not supported.

Restrictions:

None.

Example:

```
/* set environment attribute */
cliRC = SQLSetEnvAttr(henv, SQL_ATTR_OUTPUT_NTS, (SQLPOINTER) SQL_TRUE, 0);
```

Related concepts:
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "Environment attributes (CLI) list" on page 321
- "SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value" on page 176

Related samples:

- "spcall.c -- Call individual stored procedures"
- "cli_info.c -- How to get and set environment attributes at the client level"

SQLSetParam function (CLI) - Bind a parameter marker to a buffer or LOB locator

Deprecated:

Note:

In ODBC 2.0 and above, `SQLSetParam()` is deprecated and replaced with `SQLBindParameter()`.

Although this version of DB2 CLI continues to support `SQLSetParam()`, we recommend that you use `SQLBindParameter()` in your DB2 CLI programs so that they conform to the latest standards.

Equivalent function: SQLBindParameter()

The CLI function `SQLBindParameter()` is functionally the same as the `SQLSetParam()` function. Both take a similar number and type of arguments, behave the same, and return the same return codes. The difference is that `SQLSetParam()` does not have the *InputOutputType* or *BufferLength* arguments to specify the parameter type and maximum buffer length. Calling `SQLSetParam()` is functionally equivalent to calling `SQLBindParameter()` with the *InputOutputType* argument set to SQL_PARAM_INPUT and the *BufferLength* argument set to SQL_SETPARAM_VALUE_MAX.

Migrating to the new function

The statement:

```
SQLSetParam(hstmt, 1, SQL_C_SHORT, SQL_SMALLINT, 0, 0,
            &parameter1, NULL);
```

for example, would be rewritten using the new function as:

```
SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_SHORT,
                 SQL_SMALLINT, 0, 0, &parameter1,
                 SQL_SETPARAM_VALUE_MAX, NULL);
```

Related reference:

- "CLI and ODBC function summary" on page 1
- "SQLBindParameter function (CLI) - Bind a parameter marker to a buffer or LOB locator" on page 23

SQLSetPos function (CLI) - Set the cursor position in a rowset

Purpose:

Specification:	DB2 CLI 5.0	ODBC 1	

`SQLSetPos()` sets the cursor position in a rowset.

Syntax:

```
SQLRETURN   SQLSetPos (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLSETPOSIROW     RowNumber,        /* irow */
               SQLUSMALLINT      Operation,        /* fOption */
               SQLUSMALLINT      LockType);        /* fLock */
```

Function arguments:

Table 137. SQLSetPos arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle.
SQLSETPOSIROW	*RowNumber*	input	Position of the row in the rowset on which to perform the operation specified with the *Operation* argument. If *RowNumber* is 0, the operation applies to every row in the rowset. For additional information, see "RowNumber argument" on page 288.
SQLUSMALLINT	*Operation*	input	Operation to perform: • SQL_POSITION • SQL_REFRESH • SQL_UPDATE • SQL_DELETE • SQL_ADD ODBC also specifies the following operations for backwards compatibility only, which DB2 CLI also supports: • SQL_ADD While DB2 CLI does support SQL_ADD in `SQLSetPos()` calls, it is suggested that you use `SQLBulkOperations()` with the *Operation* argument set to SQL_ADD.
SQLUSMALLINT	*LockType*	input	Specifies how to lock the row after performing the operation specified in the *Operation* argument. • SQL_LOCK_NO_CHANGE ODBC also specifies the following operations which DB2 CLI does not support: • SQL_LOCK_EXCLUSIVE • SQL_LOCK_UNLOCK For additional information, see "LockType argument" on page 290.

Usage:

RowNumber argument

The *RowNumber* argument specifies the number of the row in the rowset on which to perform the operation specified by the *Operation* argument. If *RowNumber* is 0, the operation applies to every row in the rowset. *RowNumber* must be a value from 0 to the number of rows in the rowset.

Note In the C language, arrays are 0-based, while the *RowNumber* argument is 1-based. For example, to update the fifth row of the rowset, an application modifies the rowset buffers at array index 4, but specifies a *RowNumber* of 5.

All operations position the cursor on the row specified by *RowNumber*. The following operations require a cursor position:
- Positioned update and delete statements.
- Calls to `SQLGetData()`.
- Calls to `SQLSetPos()` with the SQL_DELETE, SQL_REFRESH, and SQL_UPDATE options.

An application can specify a cursor position when it calls `SQLSetPos()`. Generally, it calls `SQLSetPos()` with the SQL_POSITION or SQL_REFRESH operation to position the cursor before executing a positioned update or delete statement or calling `SQLGetData()`.

Operation argument

To determine which options are supported by a data source, an application calls `SQLGetInfo()` with one of the following information types, depending on the type of cursor:
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES1

SQL_POSITION

DB2 CLI positions the cursor on the row specified by *RowNumber*.

The contents of the row status array pointed to by the SQL_ATTR_ROW_OPERATION_PTR statement attribute are ignored for the SQL_POSITION *Operation*.

SQL_REFRESH

DB2 CLI positions the cursor on the row specified by *RowNumber* and refreshes data in the rowset buffers for that row. For more information about how DB2 CLI returns data in the rowset buffers, see the descriptions of row-wise and column-wise binding.

`SQLSetPos()` with an *Operation* of SQL_REFRESH simply updates the status and content of the rows within the current fetched rowset. This includes refreshing the bookmarks. The data in the buffers is refreshed, but not refetched, so the membership in the rowset is fixed.

A successful refresh with `SQLSetPos()` will not change a row status of SQL_ROW_DELETED. Deleted rows within the rowset will continue to be marked as deleted until the next fetch. The rows will disappear at the next fetch if the cursor supports packing (in which case a subsequent `SQLFetch()` or `SQLFetchScroll()` does not return deleted rows).

A successful refresh with SQLSetPos() will change a row status of SQL_ROW_ADDED to SQL_ROW_SUCCESS (if the row status array exists).

A refresh with SQLSetPos() will change a row status of SQL_ROW_UPDATED to the row's new status (if the row status array exists).

If an error occurs in a SQLSetPos() operation on a row, the row status is set to SQL_ROW_ERROR (if the row status array exists).

For a cursor opened with a SQL_ATTR_CONCURRENCY statement attribute of SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES, a refresh with SQLSetPos() will update the optimistic concurrency values used by the data source to detect that the row has changed. This occurs for each row that is refreshed.

The contents of the row status array are ignored for the SQL_REFRESH *Operation*.

SQL_UPDATE

DB2 CLI positions the cursor on the row specified by *RowNumber* and updates the underlying row of data with the values in the rowset buffers (the *TargetValuePtr* argument in SQLBindCol()). It retrieves the lengths of the data from the length/indicator buffers (the *StrLen_or_IndPtr* argument in SQLBindCol()). If the length of any column is SQL_COLUMN_IGNORE, the column is not updated. After updating the row, the corresponding element of the row status array is updated to SQL_ROW_UPDATED or SQL_ROW_SUCCESS_WITH_INFO (if the row status array exists).

The row operation array pointed to by the SQL_ATTR_ROW_OPERATION_PTR statement attribute can be used to indicate that a row in the current rowset should be ignored during a bulk update. For more information, see "Status and operation arrays" on page 290.

SQL_DELETE

DB2 CLI positions the cursor on the row specified by *RowNumber* and deletes the underlying row of data. It changes the corresponding element of the row status array to SQL_ROW_DELETED. After the row has been deleted, the following are not valid for the row:

- positioned update and delete statements
- calls to SQLGetData()
- calls to SQLSetPos() with *Operation* set to anything except SQL_POSITION.

Deleted rows remain visible to static and keyset-driven cursors; however, the entry in the implementation row status array (pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute) for the deleted row is changed to SQL_ROW_DELETED.

The row operation array pointed to by the SQL_ATTR_ROW_OPERATION_PTR statement attribute can be used to indicate that a row in the current rowset should be ignored during a bulk delete. For more information, see "Status and operation arrays" on page 290.

SQL_ADD

ODBC also specifies the SQL_ADD *Operation* for backwards compatibility only, which DB2 CLI also supports. It is suggested, however, that you use `SQLBulkOperations()` with the *Operation* argument set to SQL_ADD.

LockType argument

The *LockType* argument provides a way for applications to control concurrency. Generally, data sources that support concurrency levels and transactions will only support the SQL_LOCK_NO_CHANGE value of the *LockType* argument.

Although the *LockType* argument is specified for a single statement, the lock accords the same privileges to all statements on the connection. In particular, a lock that is acquired by one statement on a connection can be unlocked by a different statement on the same connection.

ODBC defines the following *LockType* arguments. DB2 CLI supports SQL_LOCK_NO_CHANGE. To determine which locks are supported by a data source, an application calls `SQLGetInfo()` with the SQL_LOCK_TYPES information type.

Table 138. Operation values

LockType argument	Lock type
SQL_LOCK_NO_CHANGE	Ensures that the row is in the same locked or unlocked state as it was before `SQLSetPos()` was called. This value of *LockType* allows data sources that do not support explicit row-level locking to use whatever locking is required by the current concurrency and transaction isolation levels.
SQL_LOCK_EXCLUSIVE	Not supported by DB2 CLI. Locks the row exclusively.
SQL_LOCK_UNLOCK	Not supported by DB2 CLI. Unlocks the row.

Status and operation arrays

The following status and operation arrays are used when calling `SQLSetPos()`:

- The row status array (as pointed to by the SQL_DESC_ARRAY_STATUS_PTR field in the IRD and the SQL_ATTR_ROW_STATUS_ARRAY statement attribute) contains status values for each row of data in the rowset. The status values are set in this array after a call to `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()`. This array is pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute.
- The row operation array (as pointed to by the SQL_DESC_ARRAY_STATUS_PTR field in the ARD and the SQL_ATTR_ROW_OPERATION_ARRAY statement attribute) contains a value for each row in the rowset that indicates whether a call to `SQLSetPos()` for a bulk operation is ignored or performed. Each element in the array is set to either SQL_ROW_PROCEED (the default) or SQL_ROW_IGNORE. This array is pointed to by the SQL_ATTR_ROW_OPERATION_PTR statement attribute.

The number of elements in the status and operation arrays must equal the number of rows in the rowset (as defined by the SQL_ATTR_ROW_ARRAY_SIZE statement attribute).

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_NEED_DATA
- SQL_STILL_EXECUTING

- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 139. SQLSetPos SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01004	Data truncated.	The *Operation* argument was SQL_REFRESH, and string or binary data returned for a column or columns with a data type of SQL_C_CHAR or SQL_C_BINARY resulted in the truncation of non-blank character or non-NULL binary data
01S01	Error in row.	The *RowNumber* argument was 0 and an error occurred in one or more rows while performing the operation specified with the *Operation* argument. (SQL_SUCCESS_WITH_INFO is returned if an error occurs on one or more, but not all, rows of a multirow operation, and SQL_ERROR is returned if an error occurs on a single-row operation.)
01S07	Fractional truncation.	The *Operation* argument was SQL_REFRESH, the data type of the application buffer was not SQL_C_CHAR or SQL_C_BINARY, and the data returned to application buffers for one or more columns was truncated. For numeric data types, the fractional part of the number was truncated. For time and timestamp data types, the fractional portion of the time was truncated.
07006	Invalid conversion.	The data value of a column in the result set could not be converted to the data type specified by *TargetType* in the call to `SQLBindCol()`.
07009	Invalid descriptor index.	The argument *Operation* was SQL_REFRESH or SQL_UPDATE and a column was bound with a column number greater than the number of columns in the result set or a column number less than 0.
21S02	Degrees of derived table does not match column list.	The argument *Operation* was SQL_UPDATE and no columns were updateable because all columns were either unbound, read-only, or the value in the bound length/indicator buffer was SQL_COLUMN_IGNORE.
22001	String data right truncation.	The assignment of a character or binary value to a column resulted in the truncation of non-blank (for characters) or non-null (for binary) characters or bytes.
22003	Numeric value out of range.	The argument *Operation* was SQL_UPDATE and the assignment of a numeric value to a column in the result set caused the whole (as opposed to fractional) part of the number to be truncated. The argument *Operation* was SQL_REFRESH, and returning the numeric value for one or more bound columns would have caused a loss of significant digits.
22007	Invalid datetime format.	The argument *Operation* was SQL_UPDATE, and the assignment of a date or timestamp value to a column in the result set caused the year, month, or day field to be out of range. The argument *Operation* was SQL_REFRESH, and returning the date or timestamp value for one or more bound columns would have caused the year, month, or day field to be out of range.

Table 139. SQLSetPos SQLSTATEs (continued)

SQLSTATE	Description	Explanation
22008	Datetime field overflow.	The *Operation* argument was SQL_UPDATE, and the performance of datetime arithmetic on data being sent to a column in the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar. The *Operation* argument was SQL_REFRESH, and the performance of datetime arithmetic on data being retrieved from the result set resulted in a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the natural rules for datetimes based on the Gregorian calendar.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The specified *StatementHandle* was not in an executed state. The function was called without first calling `SQLExecDirect()`, `SQLExecute()`, or a catalog function. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. `SQLExecute()`, `SQLExecDirect()`, or `SQLSetPos()` was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns. An ODBC 2.0 application called `SQLSetPos()` for a *StatementHandle* before `SQLFetchScroll()` was called or after `SQLFetch()` was called, and before `SQLFreeStmt()` was called with the SQL_CLOSE option.
HY011	Operation invalid at this time.	An ODBC 2.0 application set the SQL_ATTR_ROW_STATUS_PTR statement attribute; then `SQLSetPos()` was called before `SQLFetch()`, `SQLFetchScroll()`, or `SQLExtendedFetch()` was called.

Table 139. SQLSetPos SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY090	Invalid string or buffer length.	The *Operation* argument was SQL_ADD, SQL_UPDATE, or SQL_UPDATE_BY_BOOKMARK, a data value was a null pointer, and the column length value was not 0, SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. The *Operation* argument was SQL_ADD, SQL_UPDATE, or SQL_UPDATE_BY_BOOKMARK, a data value was not a null pointer, and the column length value was less than 0, but not equal to SQL_DATA_AT_EXEC, SQL_COLUMN_IGNORE, SQL_NTS, or SQL_NULL_DATA, or less than or equal to SQL_LEN_DATA_AT_EXEC_OFFSET. A value in a length/indicator buffer was SQL_DATA_AT_EXEC; the SQL type was either SQL_LONGVARCHAR, SQL_LONGVARBINARY, or a other, data-source-specific data type; and the SQL_NEED_LONG_DATA_LEN information type in `SQLGetInfo()` was "Y".
HY092	Option type out of range.	The *Operation* argument was SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK, or SQL_REFRESH_BY_BOOKMARK, and the SQL_ATTR_USE_BOOKMARKS statement attribute was set to SQL_UB_OFF.
HY107	Row value out of range.	The value specified for the argument *RowNumber* was greater than the number of rows in the rowset.
HY109	Invalid cursor position.	The cursor associated with the *StatementHandle* was defined as forward only, so the cursor could not be positioned within the rowset. See the description for the SQL_ATTR_CURSOR_TYPE attribute in `SQLSetStmtAttr()`. The *Operation* argument was SQL_UPDATE, SQL_DELETE, or SQL_REFRESH, and the row identified by the *RowNumber* argument had been deleted or had not be fetched. The *RowNumber* argument was 0 and the *Operation* argument was SQL_POSITION.
HYC00	Driver not capable.	DB2 CLI or the data source does not support the operation requested in the *Operation* argument or the *LockType* argument.
HYT00	Timeout expired	The query timeout period expired before the data source returned the result set. The timeout period is set through `SQLSetStmtAttr()` with an *Attribute* of SQL_ATTR_QUERY_TIMEOUT.

Restrictions:

None.

Example:

```
/* set the cursor position in a rowset */
cliRC = SQLSetPos(hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
```

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Result set terminology in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*

Related tasks:

- "Retrieving array data in CLI applications using column-wise binding" in *Call Level Interface Guide and Reference, Volume 1*
- "Retrieving array data in CLI applications using row-wise binding" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLBulkOperations function (CLI) - Add, update, delete or fetch a set of rows" on page 43
- "SQLFetch function (CLI) - Fetch next row" on page 118
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294

Related samples:

- "tbread.c -- How to read data from tables"

SQLSetStmtAttr function (CLI) - Set options related to a statement

Purpose:

Specification:	DB2 CLI 5.0	ODBC 3.0	ISO CLI

`SQLSetStmtAttr()` sets options related to a statement. To set an option for all statements associated with a specific connection, an application can call `SQLSetConnectAttr()`.

Refer to the CLI statement attributes list for all available statement attributes.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLSetStmtAttrW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLSetStmtAttr (
               SQLHSTMT         StatementHandle,  /* hstmt */
               SQLINTEGER       Attribute,        /* fOption */
               SQLPOINTER       ValuePtr,         /* pvParam */
               SQLINTEGER       StringLength);    /* fStrLen */
```

Function arguments:

Table 140. SQLSetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	input	Statement handle.
SQLINTEGER	*Attribute*	input	Option to set, described in the CLI statement attributes list.

Table 140. SQLSetStmtAttr arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	*ValuePtr*	input	Pointer to the value to be associated with *Attribute*. If *Attribute* is an ODBC-defined attribute, the application might need to qualify the attribute value in *ValuePtr* by setting the *StringLength* attribute as described in the *StringLength* description. If *Attribute* is a DB2 CLI attribute, the application should always qualify the attribute value in *ValuePtr* by setting the *StringLength* attribute as described in the *StringLength* description. **Note:** If *Attribute* is an ODBC attribute, *ValuePtr* can, depending on the attribute, be set to an unsigned integer. If *Attribute* is a DB2 CLI attribute, *ValuePtr* can, depending on the attribute, be set to a signed integer. If *ValuePtr* is set to a signed negative integer and an unsigned integer is expected, *ValuePtr* might be treated as a large unsigned integer by DB2 CLI without warning. Alternatively, DB2 CLI might return an error (SQLSTATE HY024).
SQLINTEGER	*StringLength*	input	If *Attribute* is an ODBC attribute, the application might need to qualify the attribute by setting *StringLength* to the following values: • If *ValuePtr* points to a character string or a binary buffer, *StringLength* should be the length of **ValuePtr*. For character string data, *StringLength* should contain the number of bytes in the string. • If *ValuePtr* is a pointer, but not to a string or binary buffer, then *StringLength* should have the value SQL_IS_POINTER. • If *ValuePtr* points to an unsigned integer, the *StringLength* attribute is ignored. If *Attribute* is a DB2 CLI attribute, the application must qualify the attribute by setting *StringLength* to the following values: • If *ValuePtr* is a pointer to a character string, then *StringLength* is the number of bytes needed to store the string or SQL_NTS. • If *ValuePtr* is a pointer to a binary buffer, then the application should place the result of the SQL_LEN_BINARY_ATTR (length) macro in *StringLength*. This places a negative value in *StringLength*. • If *ValuePtr* contains a fixed-length value, then *StringLength* is either SQL_IS_INTEGER or SQL_IS_UINTEGER, as appropriate. • If *ValuePtr* is a pointer to a value other than a character string, a binary string, or a fixed-length value, then *StringLength* should have the value SQL_IS_POINTER.

Usage:

Statement attributes for a statement remain in effect until they are changed by another call to `SQLSetStmtAttr()` or the statement is dropped by calling

SQLFreeHandle(). Calling SQLFreeStmt() with the SQL_CLOSE, SQL_UNBIND, or SQL_RESET_PARAMS options does not reset statement attributes.

Some statement attributes support substitution of a similar value if the data source does not support the value specified in **ValuePtr*. In such cases, DB2 CLI returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, DB2 CLI supports a pure keyset cursor. As a result, DB2 CLI does not allow applications to change the default value of the SQL_ATTR_KEYSET_SIZE attribute. Instead, DB2 CLI substitutes SQL_KEYSET_SIZE_DEFAULT for all other values that might be supplied in the **ValuePtr* argument and returns SQL_SUCCESS_WITH_INFO. To determine the substituted value, an application calls SQLGetStmtAttr().

The format of information set with *ValuePtr* depends on the specified *Attribute*. SQLSetStmtAttr() accepts attribute information in one of two different formats: a null-terminated character string or a 32-bit integer value. The format of information returned in SQLGetStmtAttr() reflects what was specified in SQLSetStmtAttr(). For example, character strings pointed to by the *ValuePtr* argument of SQLSetStmtAttr() have a length of *StringLength*, and this is the value that would be returned by SQLGetStmtAttr().

Setting statement attributes by setting descriptors

Many statement attributes also corresponding to a header field of one or more descriptors. These attributes can be set not only by a call to SQLSetStmtAttr(), but also by a call to SQLSetDescField(). Setting these options by a call to SQLSetStmtAttr(), rather than SQLSetDescField(), has the advantage that a descriptor handle does not have to be fetched.

Note: Calling SQLSetStmtAttr() for one statement can affect other statements. This occurs when the application parameter descriptor (APD) or application row descriptor (ARD) associated with the statement is explicitly allocated and is also associated with other statements. Because SQLSetStmtAttr() modifies the APD or ARD, the modifications apply to all statements with which this descriptor is associated. If this is not the desired behavior, the application should dissociate this descriptor from the other statement (by calling SQLSetStmtAttr() to set the SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC field to a different descriptor handle) before calling SQLSetStmtAttr() again.

When a statement attribute that is also a descriptor field is set by a call to SQLSetStmtAttr(), the corresponding field in the descriptor that is associated with the statement is also set. The field is set only for the applicable descriptors that are currently associated with the statement identified by the *StatementHandle* argument, and the attribute setting does not affect any descriptors that might be associated with that statement in the future. When a descriptor field that is also a statement attribute is set by a call to SQLSetDescField(), the corresponding statement attribute is also set.

Statement attributes determine which descriptors a statement handle is associated with. When a statement is allocated (see SQLAllocHandle()), four descriptor handles are automatically allocated and associated with the statement. Explicitly allocated descriptor handles can be associated with the statement by calling SQLAllocHandle() with a *HandleType* of SQL_HANDLE_DESC to allocate a descriptor handle, then calling SQLSetStmtAttr() to associate the descriptor handle with the statement.

The following statement attributes correspond to descriptor header fields:

Table 141. Statement attributes

Statement attribute	Header field	Descriptor
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 142. SQLSetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
01000	Warning.	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed.	DB2 CLI did not support the value specified in **ValuePtr*, or the value specified in **ValuePtr* was invalid because of SQL constraints or requirements, so DB2 CLI substituted a similar value. (Function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure.	The communication link between DB2 CLI and the data source to which it was connected failed before the function completed processing.
24000	Invalid cursor state.	The *Attribute* was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS and the cursor was open.
HY000	General error.	An error occurred for which there was no specific SQLSTATE. The error message returned by `SQLGetDiagRec()` in the **MessageText* buffer describes the error and its cause.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY009	Invalid argument value.	A null pointer was passed for *ValuePtr* and the value in **ValuePtr* was a string attribute.

Table 142. SQLSetStmtAttr SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY010	Function sequence error.	An asynchronously executing function was called for the *StatementHandle* and was still executing when this function was called. SQLExecute() or SQLExecDirect() was called for the *StatementHandle* and returned SQL_NEED_DATA. This function was called before data was sent for all data-at-execution parameters or columns.
HY011	Operation invalid at this time.	The *Attribute* was SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR, or SQL_ATTR_USE_BOOKMARKS and the statement was prepared.
HY017	Invalid use of an automatically allocated descriptor handle.	The *Attribute* argument was SQL_ATTR_IMP_ROW_DESC or SQL_ATTR_IMP_PARAM_DESC. The *Attribute* argument was SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and the value in **ValuePtr* was an implicitly allocated descriptor handle.
HY024	Invalid attribute value.	Given the specified *Attribute* value, an invalid value was specified in **ValuePtr*. (DB2 CLI returns this SQLSTATE only for connection and statement attributes that accept a discrete set of values, such as SQL_ATTR_ACCESS_MODE.)
HY090	Invalid string or buffer length.	The *StringLength* argument was less than 0, but was not SQL_NTS.
HY092	Option type out of range.	The value specified for the argument *Attribute* was not valid for this version of DB2 CLI.
HYC00	Driver not capable.	The value specified for the argument *Attribute* was a valid connection or statement attribute for the version of the DB2 CLI driver, but was not supported by the data source.

Restrictions:

None.

Example:

```
/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROW_ARRAY_SIZE,
                       (SQLPOINTER)ROWSET_SIZE,
                       0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROW_BIND_TYPE,
                       SQL_BIND_BY_COLUMN,
                       0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

/* set the required statement attributes */
cliRC = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROWS_FETCHED_PTR,
                       &rowsFetchedNb,
                       0);
STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
```

Related concepts:

- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLAllocHandle function (CLI) - Allocate handle" on page 6
- "SQLCancel function (CLI) - Cancel statement" on page 49
- "SQLFreeStmt function (CLI) - Free (or reset) a statement handle" on page 143
- "SQLGetConnectAttr function (CLI) - Get current attribute setting" on page 146
- "SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute" on page 216
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "SQLSetDescField function (CLI) - Set a single field of a descriptor record" on page 276
- "Statement attributes (CLI) list" on page 348

Related samples:

- "tbread.c -- How to read data from tables"
- "dbuse.c -- How to use a database"

SQLSetStmtOption function (CLI) - Set statement option

Deprecated:

Note:

> In ODBC 3.0, `SQLSetStmtOption()` has been deprecated and replaced with `SQLSetStmtAttr()`.
>
> Although this version of DB2 CLI continues to support `SQLSetStmtOption()`, we recommend that you use `SQLSetStmtAttr()` in your DB2 CLI programs so that they conform to the latest standards.

Note: This deprecated function cannot be used in a 64-bit environment.

Migrating to the new function

The statement:

```
SQLSetStmtOption(
          hstmt,
          SQL_ROWSET_SIZE,
          RowSetSize);
```

for example, would be rewritten using the new function as:

```
SQLSetStmtAttr(
          hstmt,
          SQL_ATTR_ROW_ARRAY_SIZE,
          (SQLPOINTER) RowSetSize,
          0);
```

Related reference:

- "CLI and ODBC function summary" on page 1
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294

SQLSpecialColumns function (CLI) - Get special (row identifier) columns

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

SQLSpecialColumns() returns unique row identifier information (primary key or unique index) for a table. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLSpecialColumnsW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLSpecialColumns(
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLUSMALLINT      IdentifierType,    /* fColType */
               SQLCHAR           *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR           *SchemaName,       /* szSchemaName */
               SQLSMALLINT       NameLength2,       /* cbSchemaName */
               SQLCHAR           *TableName,        /* szTableName */
               SQLSMALLINT       NameLength3,       /* cbTableName */
               SQLUSMALLINT      Scope,             /* fScope */
               SQLUSMALLINT      Nullable);         /* fNullable */
```

Function arguments:

Table 143. SQLSpecialColumns arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle
SQLUSMALLINT	*IdentifierType*	Input	Type of unique row identifier to return. Only the following type is supported: • SQL_BEST_ROWID Returns the optimal set of column(s) which can uniquely identify any row in the specified table. **Note:** For compatibility with ODBC applications, SQL_ROWVER is also recognized, but not supported; therefore, if SQL_ROWVER is specified, an empty result will be returned.
SQLCHAR *	*CatalogName*	Input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.

Table 143. SQLSpecialColumns arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	*NameLength1*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	Input	Schema qualifier of the specified table.
SQLSMALLINT	*NameLength2*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*TableName*	Input	Table name.
SQLSMALLINT	*NameLength3*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.
SQLUSMALLINT	*Scope*	Input	Minimum required duration for which the unique row identifier will be valid. *Scope* must be one of the following: • SQL_SCOPE_CURROW: The row identifier is guaranteed to be valid only while positioned on that row. A later re-select using the same row identifier values might not return a row if the row was updated or deleted by another transaction. • SQL_SCOPE_TRANSACTION: The row identifier is guaranteed to be valid for the duration of the current transaction. • SQL_SCOPE_SESSION: The row identifier is guaranteed to be valid for the duration of the connection. The duration over which a row identifier value is guaranteed to be valid depends on the current transaction isolation level.
SQLUSMALLINT	*Nullable*	Input	Determines whether to return special columns that can have a NULL value. Must be one of the following: • SQL_NO_NULLS - The row identifier column set returned cannot have any NULL values. • SQL_NULLABLE - The row identifier column set returned might include columns where NULL values are permitted.

Usage:

If multiple ways exist to uniquely identify any row in a table (for example, if there are multiple unique indexes on the specified table), then DB2 CLI will return the *best* set of row identifier column set based on its internal criterion.

If the schema qualifier argument associated with a table name is not specified, then the schema name defaults to the one currently in effect for the current connection.

If there is no column set which allows any row in the table to be uniquely identified, an empty result set is returned.

The unique row identifier information is returned in the form of a result set where each column of the row identifier is represented by one row in the result set. "Columns returned by SQLSpecialColumns" shows the order of the columns in the result set returned by `SQLSpecialColumns()`, sorted by SCOPE.

Since calls to `SQLSpecialColumns()` in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_COLUMN_NAME_LEN to determine the actual length of the COLUMN_NAME column supported by the connected DBMS.

Although new columns might be added and the names of the columns changed in future releases, the position of the current columns will not change.

Columns returned by SQLSpecialColumns

Column 1 SCOPE (SMALLINT)
The duration for which the name in COLUMN_NAME is guaranteed to point to the same row. Valid values are the same as for the *Scope* argument: Actual scope of the row identifier. Contains one of the following values:
- SQL_SCOPE_CURROW
- SQL_SCOPE_TRANSACTION
- SQL_SCOPE_SESSION

Refer to *Scope* in Table 143 on page 300 for a description of each value.

Column 2 COLUMN_NAME (VARCHAR(128) not NULL)
Name of the column that is (or is part of) the table's primary key.

Column 3 DATA_TYPE (SMALLINT not NULL)
SQL data type of the column.

Column 4 TYPE_NAME (VARCHAR(128) not NULL)
DBMS character string representation of the name associated with DATA_TYPE column value.

Column 5 COLUMN_SIZE (INTEGER)
If the DATA_TYPE column value denotes a character or binary string, then this column contains the maximum length in bytes; if it is a graphic (DBCS) string, this is the number of double byte characters for the parameter.

For date, time, timestamp data types, this is the total number of SQLCHAR or SQLWCHAR elements required to display the value when converted to character.

For numeric data types, this is either the total number of digits, or the total number of bits allowed in the column, depending on the value in the NUM_PREC_RADIX column in the result set.

Refer to the table of data type precision.

Column 6 BUFFER_LENGTH (INTEGER)
The maximum number of bytes for the associated C buffer to store data from this column if SQL_C_DEFAULT were specified on the `SQLBindCol()`,

SQLGetData() and SQLBindParameter() calls. This length does not include any null-terminator. For exact numeric data types, the length accounts for the decimal and the sign.

Refer to the table of data type length.

Column 7 DECIMAL_DIGITS (SMALLINT)
The scale of the column. NULL is returned for data types where scale is not applicable. Refer to the table of data type scale.

Column 8 PSEUDO_COLUMN (SMALLINT)
Indicates whether or not the column is a pseudo-column DB2 Call Level Interface will only return:

- SQL_PC_NOT_PSEUDO

DB2 DBMSs do not support pseudo columns. ODBC applications might receive the following values from other non-IBM RDBMS servers:
- SQL_PC_UNKNOWN
- SQL_PC_PSEUDO

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 144. SQLSpecialColumns SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, SQLCancel() was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	*TableName* is null.
HY010	Function sequence error.	The function was called while in a data-at-execute (SQLParamData(), SQLPutData()) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.

Table 144. SQLSpecialColumns SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the length arguments was less than 0, but not equal to SQL_NTS. The value of one of the length arguments exceeded the maximum length supported by the DBMS for that qualifier or name.
HY097	Column type out of range.	An invalid *IdentifierType* value was specified.
HY098	Scope type out of range.	An invalid *Scope* value was specified.
HY099	Nullable type out of range.	An invalid *Nullable* values was specified.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get special columns */
cliRC = SQLSpecialColumns(hstmt,
                          SQL_BEST_ROWID,
                          NULL,
                          0,
                          tbSchema,
                          SQL_NTS,
                          tbName,
                          SQL_NTS,
                          SQL_SCOPE_CURROW,
                          SQL_NULLABLE);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "Isolation levels" in *SQL Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "Data type length (CLI) table" on page 391
- "Data type precision (CLI) table" on page 389
- "Data type scale (CLI) table" on page 390
- "SQLColumns function (CLI) - Get column information for a table" on page 67
- "SQLStatistics function (CLI) - Get index and statistics information for a base table" on page 305

- "SQLTables function (CLI) - Get table information" on page 314

Related samples:

- "tbconstr.c -- How to work with constraints associated with tables"

SQLStatistics function (CLI) - Get index and statistics information for a base table

Purpose:

Specification:	DB2 CLI 2.1	ODBC 1.0	

SQLStatistics() retrieves index information for a given table. It also returns the cardinality and the number of pages associated with the table and the indexes on the table. The information is returned in a result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is SQLStatisticsW(). Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLStatistics    (
               SQLHSTMT          StatementHandle,  /* hstmt */
               SQLCHAR           *CatalogName,     /* szCatalogName */
               SQLSMALLINT       NameLength1,      /* cbCatalogName */
               SQLCHAR           *SchemaName,      /* szSchemaName */
               SQLSMALLINT       NameLength2,      /* cbSchemaName */
               SQLCHAR           *TableName,       /* szTableName */
               SQLSMALLINT       NameLength3,      /* cbTableName */
               SQLUSMALLINT      Unique,           /* fUnique */
               SQLUSMALLINT      Reserved);        /* fAccuracy */
```

Function arguments:

Table 145. SQLStatistics arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.
SQLCHAR *	*CatalogName*	Input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	Input	Schema qualifier of the specified table.
SQLSMALLINT	*NameLength2*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.

Table 145. SQLStatistics arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	*TableName*	Input	Table name.
SQLSMALLINT	*NameLength3*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.
SQLUSMALLINT	*Unique*	Input	Type of index information to return: • SQL_INDEX_UNIQUE Only unique indexes will be returned. • SQL_INDEX_ALL All indexes will be returned.
SQLUSMALLINT	*Reserved*	Input	Indicate whether the CARDINALITY and PAGES columns in the result set contain the most current information: • SQL_ENSURE : This value is reserved for future use, when the application requests the most up to date statistics information. **New applications should not use this value**. Existing applications specifying this value will receive the same results as SQL_QUICK. • SQL_QUICK : Statistics which are readily available at the server are returned. The values might not be current, and no attempt is made to ensure that they be up to date.

Usage:

`SQLStatistics()` returns two types of information:

- Statistics information for the table (if it is available):
  - when the TYPE column of the result set described below is set to SQL_TABLE_STAT, the number of rows in the table and the number of pages used to store the table.
  - when the TYPE column of the result set indicates an index, the number of unique values in the index, and the number of pages used to store the indexes.
- Information about each index, where each index column is represented by one row of the result set. The result set columns are given in "Columns returned by SQLStatistics" on page 307 in the order shown; the rows in the result set are ordered by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME and KEY_SEQ.

Since calls to `SQLStatistics()` in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

If the schema qualifier argument associated with a table name is not specified, then the schema name defaults to the one currently in effect for the current connection.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_OWNER_SCHEMA_LEN, SQL_MAX_TABLE_NAME_LEN, and

SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns returned by SQLStatistics

Column 1 TABLE_CAT (VARCHAR(128))
Catalog name of the table for which the index applies. The value is NULL if this table does not have catalogs.

Column 2 TABLE_SCHEM (VARCHAR(128))
Name of the schema containing TABLE_NAME.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the table.

Column 4 NON_UNIQUE (SMALLINT)
Indicates whether the index prohibits duplicate values:
- SQL_TRUE if the index allows duplicate values.
- SQL_FALSE if the index values must be unique.
- NULL is returned if the TYPE column indicates that this row is SQL_TABLE_STAT (statistics information on the table itself).

Column 5 INDEX_QUALIFIER (VARCHAR(128))
The string that would be used to qualify the index name in the DROP INDEX statement. Appending a period (.) plus the INDEX_NAME results in a full specification of the index.

Column 6 INDEX_NAME (VARCHAR(128))
The name of the index. If the TYPE column has the value SQL_TABLE_STAT, this column has the value NULL.

Column 7 TYPE (SMALLINT not NULL)
Indicates the type of information contained in this row of the result set:
- SQL_TABLE_STAT - Indicates this row contains statistics information on the table itself.
- SQL_INDEX_CLUSTERED - Indicates this row contains information on an index, and the index type is a clustered index.
- SQL_INDEX_HASHED - Indicates this row contains information on an index, and the index type is a hashed index.
- SQL_INDEX_OTHER - Indicates this row contains information on an index, and the index type is other than clustered or hashed.

Column 8 ORDINAL_POSITION (SMALLINT)
Ordinal position of the column within the index whose name is given in the INDEX_NAME column. A NULL value is returned for this column if the TYPE column has the value of SQL_TABLE_STAT.

Column 9 COLUMN_NAME (VARCHAR(128))
Name of the column in the index. A NULL value is returned for this column if the TYPE column has the value of SQL_TABLE_STAT.

Column 10 ASC_OR_DESC (CHAR(1))
Sort sequence for the column; "**A**" for ascending, "**D**" for descending. NULL value is returned if the value in the TYPE column is SQL_TABLE_STAT.

Column 11 CARDINALITY (INTEGER)

- If the TYPE column contains the value SQL_TABLE_STAT, this column contains the number of rows in the table.
- If the TYPE column value is not SQL_TABLE_STAT, this column contains the number of unique values in the index.
- A NULL value is returned if information is not available from the DBMS.

Column 12 PAGES (INTEGER)

- If the TYPE column contains the value SQL_TABLE_STAT, this column contains the number of pages used to store the table.
- If the TYPE column value is not SQL_TABLE_STAT, this column contains the number of pages used to store the indexes.
- A NULL value is returned if information is not available from the DBMS.

Column 13 FILTER_CONDITION (VARCHAR(128))

If the index is a filtered index, this is the filter condition. Since DB2 servers do not support filtered indexes, NULL is always returned. NULL is also returned if TYPE is SQL_TABLE_STAT.

For the row in the result set that contains table statistics (TYPE is set to SQL_TABLE_STAT), the columns values of NON_UNIQUE, INDEX_QUALIFIER, INDEX_NAME, ORDINAL_POSITION, COLUMN_NAME, and ASC_OR_DESC are set to NULL. If the CARDINALITY or PAGES information cannot be determined, then NULL is returned for those columns.

Note: An application can check the SQLERRD(3) and SQLERRD(4) fields of the SQLCA to gather some statistics on a table. However, the accuracy of the information returned in those fields depends on many factors, such as the use of parameter markers and expressions within the statement. The main factor which can be controlled is the accuracy of the database statistics. That is, when the statistics were last updated, (for example, for DB2 Database for Linux, UNIX, and Windows, the last time the `RUNSTATS` command was run). Therefore, the statistics information returned by `SQLStatistics()` is often more consistent and reliable than the statistics information contained in the SQLCA fields discussed above.

Return codes:

- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 146. SQLStatistics SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.

Table 146. SQLStatistics SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	*TableName* is null.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The valid of one of the name length arguments exceeded the maximum value supported for that data source. The maximum supported value can be obtained by calling the `SQLGetInfo()` function.
HY100	Uniqueness option type out of range.	An invalid *Unique* value was specified.
HY101	Accuracy option type out of range.	An invalid *Reserved* value was specified.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get index and statistics information for a base table */
cliRC = SQLStatistics(hstmt,
                      NULL,
                      0,
                      tbSchema,
                      SQL_NTS,
                      tbName,
                      SQL_NTS,
                      SQL_INDEX_UNIQUE,
                      SQL_QUICK);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLColumns function (CLI) - Get column information for a table" on page 67
- "SQLSpecialColumns function (CLI) - Get special (row identifier) columns" on page 300

Related samples:

- "tbinfo.c -- How to get information about tables from the system catalog tables"
- "tbconstr.c -- How to work with constraints associated with tables"

SQLTablePrivileges function (CLI) - Get privileges associated with a table

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLTablePrivileges()` returns a list of tables and associated privileges for each table. The information is returned in an SQL result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLTablePrivilegesW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN SQLTablePrivileges (
            SQLHSTMT         StatementHandle,   /* hstmt */
            SQLCHAR          *CatalogName,      /* *szCatalogName */
            SQLSMALLINT      NameLength1,       /* cbCatalogName */
            SQLCHAR          *SchemaName,       /* *szSchemaName */
            SQLSMALLINT      NameLength2,       /* cbSchemaName */
            SQLCHAR          *TableName,        /* *szTableName */
            SQLSMALLINT      NameLength3);      /* cbTableName */
```

Function arguments:

Table 147. SQLTablePrivileges arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.

Table 147. SQLTablePrivileges arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	*CatalogName*	Input	Catalog qualifier of a 3-part table name. If the target DBMS does not support 3-part naming, and *PKCatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	Input	Buffer that can contain a *pattern value* to qualify the result set by schema name.
SQLSMALLINT	*NameLength2*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*TableName*	Input	Buffer that can contain a *pattern value* to qualify the result set by table name.
SQLSMALLINT	*NameLength3*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.

Note that the *SchemaName* and *TableName* input arguments accept search patterns.

Usage:

The results are returned as a standard result set containing the columns listed in the following table. The result set is ordered by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and PRIVILEGE. If multiple privileges are associated with any given table, each privilege is returned as a separate row.

The granularity of each privilege reported here might or might not apply at the column level; for example, for some data sources, if a table can be updated, every column in that table can also be updated. For other data sources, the application must call `SQLColumnPrivileges()` to discover if the individual columns have the same table privileges.

Since calls to `SQLTablePrivileges()` in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

Sometimes, an application calls the function and no attempt is made to restrict the result set returned. For some data sources that contain a large quantity of tables, views and aliases for example, this scenario maps to an extremely large result set and very long retrieval times. In order to help reduce the long retrieval times, the configuration keyword SchemaList can be specified in the CLI initialization file to help restrict the result set when the application has supplied a null pointer for SchemaName. If the application specifies a SchemaName string, the SchemaList keyword is still used to restrict the output. Therefore, if the schema name supplied is not in the SchemaList string, then the result will be an empty result set.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call `SQLGetInfo()` with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_OWNER_SCHEMA_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns returned by SQLTablePrivileges

Column 1 TABLE_CAT (VARCHAR(128))
Catalog table name. The value is NULL if this table does not have catalogs.

Column 2 TABLE_SCHEM (VARCHAR(128))
Name of the schema contain TABLE_NAME.

Column 3 TABLE_NAME (VARCHAR(128) not NULL)
Name of the table.

Column 4 GRANTOR (VARCHAR(128))
Authorization ID of the user who granted the privilege.

Column 5 GRANTEE (VARCHAR(128))
Authorization ID of the user to whom the privilege is granted.

Column 6 PRIVILEGE (VARCHAR(128))
Table privilege. This can be one of the following strings:
- ALTER
- CONTROL
- INDEX
- DELETE
- INSERT
- REFERENCES
- SELECT
- UPDATE

Column 7 IS_GRANTABLE (VARCHAR(3))
Indicates whether the grantee is permitted to grant the privilege to other users.

This can be ″YES″, ″NO″ or NULL.

Note: The column names used by DB2 CLI follow the X/Open CLI CAE specification style. The column types, contents and order are identical to those defined for the `SQLProcedures()` result set in ODBC.

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 148. SQLTablePrivileges SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The valid of one of the name length arguments exceeded the maximum value supported for that data source. The maximum supported value can be obtained by calling the `SQLGetInfo()` function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get privileges associated with a table */
cliRC = SQLTablePrivileges(hstmt,
                           NULL,
                           0,
                           tbSchemaPattern,
                           SQL_NTS,
                           tbNamePattern,
                           SQL_NTS);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLProcedures function (CLI) - Get list of procedure names" on page 257
- "SQLTables function (CLI) - Get table information" on page 314
- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SchemaList CLI/ODBC configuration keyword" in *Call Level Interface Guide and Reference, Volume 1*

Related samples:

- "tbinfo.c -- How to get information about tables from the system catalog tables"

SQLTables function (CLI) - Get table information

Purpose:

Specification:	**DB2 CLI** 2.1	**ODBC** 1.0	

`SQLTables()` returns a list of table names and associated information stored in the system catalog of the connected data source. The list of table names is returned as a result set, which can be retrieved using the same functions that are used to process a result set generated by a query.

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLTablesW()`. Refer to Unicode functions (CLI) for information on ANSI to Unicode function mappings.

Syntax:

```
SQLRETURN   SQLTables        (
               SQLHSTMT          StatementHandle,   /* hstmt */
               SQLCHAR           *CatalogName,      /* szCatalogName */
               SQLSMALLINT       NameLength1,       /* cbCatalogName */
               SQLCHAR           *SchemaName,       /* szSchemaName */
               SQLSMALLINT       NameLength2,       /* cbSchemaName */
               SQLCHAR           *TableName,        /* szTableName */
               SQLSMALLINT       NameLength3,       /* cbTableName */
               SQLCHAR           *TableType,        /* szTableType */
               SQLSMALLINT       NameLength4);      /* cbTableType */
```

Function arguments:

Table 149. SQLTables arguments

Data type	Argument	Use	Description
SQLHSTMT	*StatementHandle*	Input	Statement handle.

Table 149. SQLTables arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	*CatalogName*	Input	Catalog qualifier of a 3-part table name that can contain a *pattern value*. If the target DBMS does not support 3-part naming, and *CatalogName* is not a null pointer and does not point to a zero-length string, then an empty result set and SQL_SUCCESS will be returned. Otherwise, this is a valid filter for DBMSs that support 3-part naming.
SQLSMALLINT	*NameLength1*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *CatalogName*, or SQL_NTS if *CatalogName* is null-terminated.
SQLCHAR *	*SchemaName*	Input	Buffer that can contain a *pattern value* to qualify the result set by schema name.
SQLSMALLINT	*NameLength2*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *SchemaName*, or SQL_NTS if *SchemaName* is null-terminated.
SQLCHAR *	*TableName*	Input	Buffer that can contain a *pattern value* to qualify the result set by table name.
SQLSMALLINT	*NameLength3*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableName*, or SQL_NTS if *TableName* is null-terminated.
SQLCHAR *	*TableType*	Input	Buffer that can contain a *value list* to qualify the result set by table type. The value list is a list of uppercase comma-separated single values for the table types of interest. Valid table type identifiers include: ALIAS, HIERARCHY TABLE, INOPERATIVE VIEW, NICKNAME, MATERIALIZED QUERY TABLE, SYSTEM TABLE, TABLE, TYPED TABLE, TYPED VIEW, or VIEW. If *TableType* argument is a NULL pointer or a zero length string, then this is equivalent to specifying all of the possibilities for the table type identifier. If SYSTEM TABLE is specified, then both system tables and system views (if there are any) are returned.
SQLSMALLINT	*NameLength4*	Input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store *TableType*, or SQL_NTS if *TableType* is null-terminated.

Note that the *CatalogName*, *SchemaName*, and *TableName* input arguments accept search patterns.

Usage:

Table information is returned in a result set where each table is represented by one row of the result set. To determine the type of access permitted on any given table

in the list, the application can call SQLTablePrivileges(). The application must be able to handle a situation where the user selects a table for which SELECT privileges are not granted.

To support obtaining just a list of schemas, the following special semantics for the *SchemaName* argument can be applied: if *SchemaName* is a string containing a single percent (%) character, and *CatalogName* and *TableName* are empty strings, then the result set contains a list of valid schemas in the data source.

If *TableType* is a single percent character (%) and *CatalogName, SchemaName*, and *TableName* are empty strings, then the result set contains a list of valid table types for the data source. (All columns except the TABLE_TYPE column contain NULLs.)

If *TableType* is not an empty string, it must contain a list of uppercase, comma-separated values for the types of interest; each value can be enclosed in single quotation marks or unquoted. For example, "'TABLE','VIEW'" or "TABLE,VIEW". If the data source does not support or does not recognize a specified table type, nothing is returned for that type.

Sometimes, an application calls SQLTables() with null pointers for some or all of the *SchemaName, TableName*, and *TableType* arguments so that no attempt is made to restrict the result set returned. For some data sources that contain a large quantity of tables, views and aliases for example, this scenario maps to an extremely large result set and very long retrieval times. Three mechanisms are introduced to help the user reduce the long retrieval times: three configuration keywords (SCHEMALIST, SYSSCHEMA, TABLETYPE) can be specified in the CLI initialization file to help restrict the result set when the application has supplied null pointers for either or both of *SchemaName* and *TableType*. If the application specifies a *SchemaName* string, the SCHEMALIST keyword is still used to restrict the output. Therefore, if the schema name supplied is not in the SCHEMALIST string, then the result will be an empty result set.

The result set returned by SQLTables() contains the columns listed in "Columns returned by SQLTables" in the order given. The rows are ordered by TABLE_TYPE, TABLE_CAT, TABLE_SCHEM, and TABLE_NAME.

Since calls to SQLTables() in many cases map to a complex and thus expensive query against the system catalog, they should be used sparingly, and the results saved rather than repeating calls.

The VARCHAR columns of the catalog functions result set have been declared with a maximum length attribute of 128 to be consistent with SQL92 limits. Since DB2 names are less than 128, the application can choose to always set aside room for 128 characters (plus the null-terminator) for the output buffer, or alternatively, call SQLGetInfo() with the SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_OWNER_SCHEMA_LEN, SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN to determine respectively the actual lengths of the TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME columns supported by the connected DBMS.

Although new columns might be added and the names of the existing columns changed in future releases, the position of the current columns will not change.

Columns returned by SQLTables

Column 1 TABLE_CAT (VARCHAR(128))
Name of the catalog containing TABLE_SCHEM. The value is NULL if this table does not have catalogs.

Column 2 TABLE_SCHEM (VARCHAR(128))
Name of the schema containing TABLE_NAME.

Column 3 TABLE_NAME (VARCHAR(128))
Name of the table, view, alias or synonym.

Column 4 TABLE_TYPE (VARCHAR(128))
Identifies the type given by the name in the TABLE_NAME column. It can have the string values 'ALIAS', 'HIERARCHY TABLE', 'INOPERATIVE VIEW', 'NICKNAME', 'MATERIALIZED QUERY TABLE', 'SYSTEM TABLE', 'TABLE', 'TYPED TABLE', 'TYPED VIEW', or 'VIEW'.

Column 5 REMARKS (VARCHAR(254))
Descriptive information about the table.

Column

Return codes:
- SQL_SUCCESS
- SQL_SUCCESS_WITH_INFO
- SQL_STILL_EXECUTING
- SQL_ERROR
- SQL_INVALID_HANDLE

Diagnostics:

Table 150. SQLTables SQLSTATEs

SQLSTATE	Description	Explanation
24000	Invalid cursor state.	A cursor was already opened on the statement handle.
40003 **08**S01	Communication link failure.	The communication link between the application and data source failed before the function completed.
HY001	Memory allocation failure.	DB2 CLI is unable to allocate memory required to support execution or completion of the function. It is likely that process-level memory has been exhausted for the application process. Consult the operating system configuration for information on process-level memory limitations.
HY008	Operation was cancelled.	Asynchronous processing was enabled for *StatementHandle*. The function was called and before it completed execution, `SQLCancel()` was called on *StatementHandle* from a different thread in a multithreaded application. Then the function was called again on *StatementHandle*.
HY009	Invalid argument value.	*TableName* is null.
HY010	Function sequence error.	The function was called while in a data-at-execute (`SQLParamData()`, `SQLPutData()`) operation. The function was called while within a BEGIN COMPOUND and END COMPOUND SQL operation. An asynchronously executing function (not this one) was called for the *StatementHandle* and was still executing when this function was called. The function was called before a statement was prepared on the statement handle.

Table 150. SQLTables SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY014	No more handles.	DB2 CLI was unable to allocate a handle due to resource limitations.
HY090	Invalid string or buffer length.	The value of one of the name length arguments was less than 0, but not equal to SQL_NTS. The valid of one of the name length arguments exceeded the maximum value supported for that data source. The maximum supported value can be obtained by calling the `SQLGetInfo()` function.
HYT00	Timeout expired.	The timeout period expired before the data source returned the result set. The timeout period can be set using the SQL_ATTR_QUERY_TIMEOUT attribute for `SQLSetStmtAttr()`.

Restrictions:

None.

Example:

```
/* get table information */
cliRC = SQLTables(hstmt,
                  NULL,
                  0,
                  tbSchemaPattern,
                  SQL_NTS,
                  tbNamePattern,
                  SQL_NTS,
                  NULL,
                  0);
```

Related concepts:

- "Catalog functions for querying system catalog information in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Input arguments on catalog functions in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLSTATES for DB2 CLI" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI function return codes" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLColumns function (CLI) - Get column information for a table" on page 67
- "SQLTablePrivileges function (CLI) - Get privileges associated with a table" on page 310

Related samples:

- "tbinfo.c -- How to get information about tables from the system catalog tables"
- "tbread.c -- How to read data from tables"

SQLTransact function (CLI) - Transaction management

Deprecated:

Note:

In ODBC 3.0, `SQLTransact()` has been deprecated and replaced with `SQLEndTran()`.

Although this version of DB2 CLI continues to support `SQLTransact()`, we recommend that you use `SQLEndTran()` in your DB2 CLI programs so that they conform to the latest standards.

Migrating to the new function

The statement:

```
SQLTransact(henv, hdbc, SQL_COMMIT);
```

for example, would be rewritten using the new function as:

```
SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
```

Related reference:

- "CLI and ODBC function summary" on page 1
- "SQLEndTran function (CLI) - End transactions of a connection or an Environment" on page 97

Chapter 2. CLI attributes - environment, connection, and statement

Environments, connections, and statements each have a defined set of attributes that affect how DB2 CLI behaves. These attributes have default values, however, you can modify the default DB2 CLI behavior by setting these attributes to different values. This chapter lists the environment, connection, and statement attributes that you can set to customize DB2 CLI behavior.

Environment attributes (CLI) list

Note: ODBC does not support setting driver-specific environment attributes using `SQLSetEnvAttr()`. Only CLI applications can set the DB2 CLI-specific environment attributes using this function.

SQL_ATTR_CONNECTION_POOLING
This attribute has been deprecated in DB2 Universal Database (DB2 UDB) Version 8.

SQL_ATTR_CONNECTTYPE

Note: This attribute replaces SQL_CONNECTTYPE.
A 32-bit integer value that specifies whether this application is to operate in a coordinated or uncoordinated distributed environment. . The possible values are:

- **SQL_CONCURRENT_TRANS**: The application can have concurrent multiple connections to any one database or to multiple databases. Each connection has its own commit scope. No effort is made to enforce coordination of transaction. If an application issues a commit using the environment handle on `SQLEndTran()` and not all of the connections commit successfully, the application is responsible for recovery. This is the default.
- SQL_COORDINATED_TRANS: The application wishes to have commit and rollbacks coordinated among multiple database connections. This option setting corresponds to the specification of the Type 2 CONNECT in embedded SQL. In contrast to the SQL_CONCURRENT_TRANS setting described above, the application is permitted only one open connection per database.

  Note: This connection type results in the default for SQL_ATTR_AUTOCOMMIT connection option to be SQL_AUTOCOMMIT_OFF.

If changing this attribute from the default then it must be set before any connections have been established on the environment handle.

It is recommended that the application set this attribute as an environment attribute with a call to `SQLSetEnvAttr()`, if necessary, as soon as the environment handle has been allocated. However, since ODBC applications cannot access `SQLSetEnvAttr()`, they must set this attribute using

SQLSetConnectAttr() after each connection handle is allocated, but before any connections have been established.

All connections on an environment handle must have the same SQL_ATTR_CONNECTTYPE setting. An environment cannot have a mixture of concurrent and coordinated connections. The type of the first connection will determine the type of all subsequent connections. SQLSetEnvAttr() will return an error if an application attempts to change the connection type while there is an active connection.

The default connect type can also be set using the ConnectType CLI/ODBC configuration keyword.

Note: This is an IBM defined extension.

SQL_ATTR_CP_MATCH
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_DIAGLEVEL

Description
A 32-bit integer value which represents the diagnostic level. This is equivalent to the database manager parameter DIAGLEVEL.

Values
Valid values are: 0, 1, 2, 3, or 4. (The default value is 3.)

See: diaglevel - Diagnostic error capture level configuration parameter for a details about these values.

Usage notes
This attribute must be set before any connection handles have been created.

SQL_ATTR_DIAGPATH

Description
A pointer to a null-terminated character string containing the name of the directory where diagnostic data is to be placed. This is equivalent to the database manager parameter DIAGPATH.

Values
The default value is the db2dump directory on UNIX and Linux operating systems, and the db2 directory on Windows operating systems.

Usage notes
This attribute must be set before any connection handles have been created.

SQL_ATTR_INFO_ACCTSTR
A pointer to a null-terminated character string used to identify the client accounting string sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 200 characters.

- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_APPLNAME
A pointer to a null-terminated character string used to identify the client application name sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 32 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_USERID
A pointer to a null-terminated character string used to identify the client user ID sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 16 characters.
- This user-id is not to be confused with the authentication user-id. This user-id is for identification purposes only and is not used for any authorization.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_WRKSTNNAME
A pointer to a null-terminated character string used to identify the client workstation name sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 18 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_MAXCONN
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_NOTIFY_LEVEL

Description
A 32-bit integer value which represents the notification level. This is equivalent to the database manager parameter `NOTIFYLEVEL`.

Values
Valid values are: 0, 1, 2, 3, or 4. (The default value is 3.)

See: notifylevel - Notify level configuration parameter for a details about these values.

Usage notes
This attribute must be set before any connection handles have been created.

SQL_ATTR_ODBC_VERSION
A 32-bit integer that determines whether certain functionality exhibits ODBC 2.x (DB2 CLI v2) behavior or ODBC 3.0 (DB2 CLI v5) behavior.

It is recommended that all DB2 CLI applications set this environment attribute. ODBC applications must set this environment attribute before calling any function that has an SQLHENV argument, or the call will return SQLSTATE HY010 (Function sequence error.).

The following values are used to set the value of this attribute:

- SQL_OV_ODBC3: Causes the following ODBC 3.0 (DB2 CLI v5) behavior:
  - DB2 CLI returns and expects ODBC 3.0 (DB2 CLI v5) codes for date, time, and timestamp.
  - DB2 CLI returns ODBC 3.0 (DB2 CLI v5) SQLSTATE codes when `SQLError()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` are called.
  - The *CatalogName* argument in a call to `SQLTables()` accepts a search pattern.
- SQL_OV_ODBC2 Causes the following ODBC 2.x (DB2 CLI v2) behavior:
  - DB2 CLI returns and expects ODBC 2.x (DB2 CLI v2) codes for date, time, and timestamp.
  - DB2 CLI returns ODBC 2.0 (DB2 CLI v2) SQLSTATE codes when `SQLError()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` are called.
  - The *CatalogName* argument in a call to `SQLTables()` does not accept a search pattern.

SQL_ATTR_OUTPUT_NTS
A 32-bit integer value which controls the use of null-termination in output arguments. The possible values are:

- **SQL_TRUE**: DB2 CLI uses null termination to indicate the length of output character strings (default).

  This is the default.
- SQL_FALSE: DB2 CLI does not use null termination in output character strings.

The CLI functions affected by this attribute are all functions called for the environment (and for any connections and statements allocated under the environment) that have character string parameters.

This attribute can only be set when there are no connection handles allocated under this environment.

SQL_ATTR_PROCESSCTL
A 32-bit mask that sets process level attributes which affect all environments and connections for the process. This attribute must be set before the environment handle is allocated.

The call to `SQLSetEnvAttr()` must have the *EnvironmentHandle* argument set to SQL_NULL_HANDLE. The settings remain in effect for the life of the process. Generally this attribute is only used for performance sensitive applications, where large numbers of CLI function calls are being made. Before setting any of these bits, ensure that the application, and any other libraries that the application calls, comply with the restrictions listed.

The following values can be combined to form a bitmask:

- SQL_PROCESSCTL_NOTHREAD - This bit indicates that the application does not use multiple threads, or if it does use multiple threads, guarantees that all DB2 calls will be serialized by the application. If set, DB2 CLI does not make any system calls to serialize calls to CLI, and sets the DB2 context type to SQL_CTX_ORIGINAL.
- SQL_PROCESSCTL_NOFORK - This bit indicates that the application will never fork a child process. By default, DB2 CLI does not check to see if an application forks a child process. However, if the CheckForFork CLI/ODBC configuration keyword is set, DB2 CLI checks the current process id for each function call for all applications connecting to the database for which the keyword is enabled. This attribute can be set so that DB2 CLI does not check for forked processes for that application.

Note: This is an IBM defined extension.

SQL_ATTR_SYNC_POINT
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_TRACE
A pointer to a null-terminated character string used to turn on the DB2 CLI/ODBC trace facility. The string must include the keywords TRACE and TRACEPATHNAME. For example:

```
"TRACE=1; TRACEPATHNAME=<dir>;"
```

SQL_ATTR_USE_2BYTES_OCTET_LENGTH
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA
Setting this attribute is equivalent to setting the connection attribute SQL_ATTR_DESCRIBE_OUTPUT_LEVEL to 0. SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA has been deprecated and applications should now use the connection attribute SQL_ATTR_DESCRIBE_OUTPUT_LEVEL.

SQL_ATTR_USER_REGISTRY_NAME
This attribute is only used when authenticating a user on a server that is using an identity mapping service. It is set to a user defined string that names an identity mapping registry. The format of the name varies depending on the identity mapping service. By providing this attribute you tell the server that the user name provided can be found in this registry.

After setting this attribute the value will be used on subsequent attempts to establish a normal connection, establish a trusted connection, or switch the user id on a trusted connection.

SQL_CONNECTTYPE
This *Attribute* has been replaced with "SQL_ATTR_CONNECTTYPE" on page 321.

SQL_MAXCONN
This *Attribute* has been replaced with "SQL_ATTR_MAXCONN" on page 324.

SQL_SYNC_POINT
This *Attribute* has been replaced with "SQL_ATTR_SYNC_POINT" on page 325.

Related reference:

- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*
- "Connection attributes (CLI) list" on page 326
- "SQLColAttribute function (CLI) - Return a column attribute" on page 54
- "SQLDescribeCol function (CLI) - Return a set of attributes for a column" on page 82
- "SQLGetData function (CLI) - Get data from a column" on page 152
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" on page 160
- "SQLGetEnvAttr function (CLI) - Retrieve current environment attribute value" on page 176

Connection attributes (CLI) list

The following table indicates when each of the CLI connection attributes can be set. A ″Yes″ in the ″After statements allocated″ column means that the connection attribute can be set both before and after the statements are allocated.

Table 151. When connection attributes can be set

Attribute	Before connection	After connection	After statements allocated
"SQL_ATTR_ACCESS_MODE" on page 328	Yes	Yes	Yes [a]
"SQL_ATTR_ANSI_APP" on page 328	Yes	No	No
"SQL_ATTR_APP_USES_LOB_LOCATOR" on page 328	Yes	Yes	Yes[c]
"SQL_ATTR_APPEND_FOR_FETCH_ONLY" on page 328	Yes	Yes	No
"SQL_ATTR_ASYNC_ENABLE" on page 329	Yes	Yes	Yes[a]
"SQL_ATTR_AUTO_IPD" on page 329 (read-only)	No	No	No
"SQL_ATTR_AUTOCOMMIT" on page 330	Yes	Yes	Yes [b]
"SQL_ATTR_CONN_CONTEXT" on page 330	Yes	No	No
"SQL_ATTR_CONNECT_NODE" on page 331	Yes	No	No
"SQL_ATTR_CONNECTION_DEAD" on page 331 (read-only)	No	No	No
"SQL_ATTR_CONNECTTYPE" on page 331	Yes	No	No
"SQL_ATTR_CURRENT_CATALOG" on page 332 (read-only)	No	No	No
"SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION" on page 332	Yes	Yes	Yes
"SQL_ATTR_CURRENT_PACKAGE_PATH" on page 332	Yes	Yes	Yes
"SQL_ATTR_CURRENT_PACKAGE_SET" on page 332	Yes	Yes [a]	No [*]
"SQL_ATTR_CURRENT_SCHEMA" on page 333	Yes	Yes	Yes
"SQL_ATTR_DB2_APPLICATION_HANDLE" on page 333 (read-only)	No	No	No

Table 151. When connection attributes can be set (continued)

Attribute	Before connection	After connection	After statements allocated
"SQL_ATTR_DB2_APPLICATION_ID" on page 333 (read-only)	No	No	No
"SQL_ATTR_DB2_SQLERRP" on page 333 (read-only)	No	No	No
"SQL_ATTR_DB2EXPLAIN" on page 334	No	Yes	Yes
"SQL_ATTR_DECFLOAT_ROUNDING_MODE" on page 334	Yes	Yes	Yes
"SQL_ATTR_DESCRIBE_CALL" on page 335	Yes	Yes	Yes[a]
"SQL_ATTR_DESCRIBE_OUTPUT_LEVEL" on page 335	Yes	Yes	No
"SQL_ATTR_ENLIST_IN_DTC" on page 336	No	Yes	Yes
"SQL_ATTR_FREE_LOCATORS_ON_FETCH" on page 337	Yes	Yes	Yes
"SQL_ATTR_INFO_ACCTSTR" on page 337	No	Yes	Yes
"SQL_ATTR_INFO_APPLNAME" on page 337	No	Yes	Yes
"SQL_ATTR_INFO_PROGRAMID" on page 337	No	Yes	Yes [a]
"SQL_ATTR_INFO_PROGRAMNAME" on page 338	Yes	No	No
"SQL_ATTR_INFO_USERID" on page 338	No	Yes	Yes
"SQL_ATTR_INFO_WRKSTNNAME" on page 338	No	Yes	Yes
"SQL_ATTR_KEEP_DYNAMIC" on page 338	No	Yes	Yes
"SQL_ATTR_LOB_CACHE_SIZE" on page 339	Yes	Yes	Yes[c]
"SQL_ATTR_LOGIN_TIMEOUT" on page 339	Yes	No	No
"SQL_ATTR_LONGDATA_COMPAT" on page 339	Yes	Yes	Yes
"SQL_ATTR_MAX_LOB_BLOCK_SIZE" on page 340	Yes	Yes	Yes[c]
"SQL_ATTR_MAPCHAR" on page 339	Yes	Yes	Yes
"SQL_ATTR_PING_DB" on page 340 (read only)	No	No	No
"SQL_ATTR_QUIET_MODE" on page 340	Yes	Yes	Yes
"SQL_ATTR_RECEIVE_TIMEOUT" on page 341	Yes	Yes	Yes
"SQL_ATTR_REOPT" on page 341	No	Yes	Yes[c]
"SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB" on page 342	Yes	Yes	Yes
"SQL_ATTR_SERVER_MSGTXT_MASK" on page 342	Yes	Yes	Yes
"SQL_ATTR_SERVER_MSGTXT_SP" on page 343	Yes	Yes	Yes
"SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB" on page 343	Yes	Yes	Yes
"SQL_ATTR_STREAM_GETDATA" on page 343	Yes	Yes	Yes[c]
"SQL_ATTR_TRUSTED_CONTEXT_PASSWORD" on page 344	No	Yes	Yes
"SQL_ATTR_TRUSTED_CONTEXT_USERID" on page 344	No	Yes	Yes
"SQL_ATTR_TXN_ISOLATION" on page 345	No	Yes [b]	Yes [a]
"SQL_ATTR_USE_TRUSTED_CONTEXT" on page 345	Yes	No	No
"SQL_ATTR_USER_REGISTRY_NAME" on page 346	Yes	No	No
"SQL_ATTR_WCHARTYPE" on page 346	Yes	Yes [b]	Yes [b]
"SQL_ATTR_XML_DECLARATION" on page 346	Yes	Yes	Yes [a]

[a] Will only affect subsequently allocated statements.

[b] Attribute can be set only if there are no open transactions on the connection.

[c] Attribute can be set only if there are no open cursors on the connection. The attribute will affect all statements.

* Setting this attribute after statements have been allocated will not result in an error, however, determining which packages are used by which statements is ambiguous and unexpected behavior might occur. It is not recommended that you set this attribute after statements have been allocated.

Attribute

ValuePtr contents

SQL_ATTR_ACCESS_MODE

A 32-bit integer value which can be either:

- SQL_MODE_READ_ONLY: the application is indicating that it will not be performing any updates on data from this point on. Therefore, a less restrictive isolation level and locking can be used on transactions: uncommitted read (SQL_TXN_READ_UNCOMMITTED). DB2 CLI does not ensure that requests to the database are *read-only*. If an update request is issued, DB2 CLI will process it using the transaction isolation level it has selected as a result of the SQL_MODE_READ_ONLY setting.
- **SQL_MODE_READ_WRITE (default)**: the application is indicating that it will be making updates on data from this point on. DB2 CLI will go back to using the default transaction isolation level for this connection.

There must not be any outstanding transactions on this connection.

SQL_ATTR_ANSI_APP

A 32-bit unsigned integer that identifies an application as an ANSI or Unicode application. This attribute has either of the following values:

- **SQL_AA_TRUE (default)**: the application is an ANSI application. All character data is passed to and from the application in the native application (client) codepage using the ANSI version of the CLI/ODBC functions.
- SQL_AA_FALSE: the application is a Unicode application. All character data is passed to and from the application in Unicode when the Unicode (W) versions of the CLI/ODBC functions are called.

SQL_ATTR_APP_USES_LOB_LOCATOR

A 32-bit unsigned integer that indicates if applications are using LOB locators. This attribute has either of the following values:

- **1 (default)**: Indicates that applications are using LOB locators.
- 0: For applications that do not use LOB locators and are querying data on a server that supports Dynamic Data Format, specify 0 to indicate that LOB locators are not used and allow the return of LOB data to be optimized.

This keyword is ignored for stored procedure result sets.

If the keyword is set to 0 and an application binds a LOB locator to a result set using `SQLBindCol()`, an Invalid conversion error will be returned by the `SQLFetch()` function.

Setting the AppUsesLOBLocator CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_APPEND_FOR_FETCH_ONLY

By default, DB2 CLI appends the "FOR FETCH ONLY" clause to read SELECT statements when connected to DB2 for z/OS or DB2 Universal Database for iSeries (DB2 UDB for iSeries) databases.

This attribute allows an application to control at a connection level when DB2 CLI appends the "FOR FETCH ONLY" clause. For example, an application is binding the DB2 CLI packages using different bind BLOCKING options (for example, BLOCKING UNAMBIG) and wants to suppress the blocking in order to keep positioned on a given row.

To change the default DB2 CLI behavior, the keyword is set as follows:

- 0: DB2 CLI never appends the "FOR FETCH ONLY" clause to read SELECT statements regardless of the server type it is connecting to.

- 1: DB2 CLI always appends the ″FOR FETCH ONLY″ clause to read SELECT statements regardless of the server type it is connecting to.

The attribute should be set either after the connection is allocated or immediately after it is established and should be set once for the duration of the execution of the application. Application can query the attribute with `SQLGetConnectAttr()` after connection is established or after this attribute is set.

Setting the AppendForFetchOnly CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_ASYNC_ENABLE

A 32-bit integer value that specifies whether a function called with a statement on the specified connection is executed asynchronously:

- **SQL_ASYNC_ENABLE_OFF (default)** = Off
- SQL_ASYNC_ENABLE_ON = On

Setting SQL_ASYNC_ENABLE_ON enables asynchronous execution for all statement handles allocated on this connection. An error is returned if asynchronous execution is turned on while there is an active statement on the connection.

This attribute can be set whether `SQLGetInfo()`, called with the *InfoType* SQL_ASYNC_MODE, returns SQL_AM_CONNECTION or SQL_AM_STATEMENT.

Once a function has been called asynchronously, only the original function, `SQLAllocHandle()`, `SQLCancel()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` can be called on the statement or the connection associated with *StatementHandle*, until the original function returns a code other than SQL_STILL_EXECUTING. Any other function called on *StatementHandle* or the connection associated with *StatementHandle* returns SQL_ERROR with an SQLSTATE of HY010 (Function sequence error).

The following functions can be executed asynchronously: `SQLBulkOperations()`, `SQLColAttribute()`, `SQLColumnPrivileges()`, `SQLColumns()`, `SQLDescribeCol()`, `SQLDescribeParam()`, `SQLExecDirect()`, `SQLExecute()`, `SQLExtendedFetch()`, `SQLExtendedPrepare()`, `SQLFetch()`, `SQLFetchScroll()`, `SQLForeignKeys()`, `SQLGetData()`, `SQLGetLength()`, `SQLGetPosition()`, `SQLMoreResults()`, `SQLNumResultCols()`, `SQLParamData()`, `SQLPrepare()`, `SQLPrimaryKeys()`, `SQLProcedureColumns()`, `SQLProcedures()`, `SQLRowCount()`, `SQLSetPos()`, `SQLSpecialColumns()`, `SQLStatistics()`, `SQLTablePrivileges()`, `SQLTables()`.

Note: Any Unicode equivalent of a function stated above can be called asynchronously.

SQL_ATTR_AUTO_IPD

A read-only 32-bit unsigned integer value that specifies whether automatic population of the IPD after a call to `SQLPrepare()` is supported:

- SQL_TRUE = Automatic population of the IPD after a call to `SQLPrepare()` is supported by the server.
- SQL_FALSE = Automatic population of the IPD after a call to `SQLPrepare()` is not supported by the server. Servers that do not support prepared statements will not be able to populate the IPD automatically.

If SQL_TRUE is returned for the SQL_ATTR_AUTO_IPD connection attribute, the statement attribute SQL_ATTR_ENABLE_AUTO_IPD can be set to turn automatic population of the IPD on or off. If SQL_ATTR_AUTO_IPD is SQL_FALSE, SQL_ATTR_ENABLE_AUTO_IPD cannot be set to SQL_TRUE.

The default value of SQL_ATTR_ENABLE_AUTO_IPD is equal to the value of SQL_ATTR_AUTO_IPD.

This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`.

SQL_ATTR_AUTOCOMMIT

A 32-bit unsigned integer value that specifies whether to use auto-commit or manual commit mode:

- SQL_AUTOCOMMIT_OFF: the application must manually, explicitly commit or rollback transactions with `SQLEndTran()` calls.
- **SQL_AUTOCOMMIT_ON (default)**: DB2 CLI operates in auto-commit mode by default. Each statement is implicitly committed. Each statement that is not a query is committed immediately after it has been executed or rolled back if failure occurred. Each query is committed immediately after the associated cursor is closed.

  Note: If this is a coordinated distributed unit of work connection, then the default is **SQL_AUTOCOMMIT_OFF**

Since in many DB2 environments, the execution of the SQL statements and the commit might be flowed separately to the database server, autocommit can be expensive. It is recommended that the application developer take this into consideration when selecting the auto-commit mode.

Note: Changing from manual commit to auto-commit mode will commit any open transaction on the connection.

SQL_ATTR_CLIENT_LOB_BUFFERING

Specifies whether LOB locators or the underlying LOB data is returned in a result set for LOB columns that are not bound. By default, locators are returned. If an application usually fetches unbound LOBs and then must retrieve the underlying LOB data, the application's performance can be improved by retrieving the LOB data from the outset; this reduces the number of synchronous waits and network flows. The possible values for this attribute are:

- SQL_CLIENTLOB_USE_LOCATORS (default) - LOB locators are returned
- SQL_CLIENTLOB_BUFFER_UNBOUND_LOBS - actual LOB data is returned

SQL_ATTR_CONN_CONTEXT

Indicates which context the connection should use. An SQLPOINTER to either:

- a valid context (allocated by the `sqleBeginCtx()` DB2 API) to set the context
- a NULL pointer to reset the context

This attribute can only be used when the application is using the DB2 context APIs to manage multi-threaded applications. By default, DB2 CLI

manages contexts by allocating one context per connection handle, and ensuring that any executing thread is attached to the correct context.

For more information about contexts, refer to the `sqleBeginCtx()` API.

SQL_ATTR_CONNECT_NODE
A 32-bit integer that specifies the target logical partition of a DB2 Enterprise Server Edition database partition server that you want to connect to. This setting overrides the value of the environment variable DB2NODE. It can be set to:

- an integer between 0 and 999
- SQL_CONN_CATALOG_NODE

If this variable is not set, the target logical node defaults to the logical node which is defined with port 0 on the machine.

There is also a corresponding keyword, the ConnectNode CLI/ODBC configuration keyword.

SQL_ATTR_CONNECTION_DEAD
A read only 32-bit integer value that indicates whether or not the connection is still active. DB2 CLI will return one of the following values:

- SQL_CD_FALSE - the connection is still active.
- SQL_CD_TRUE - an error has already happened and caused the connection to the server to be terminated. The application should still perform a disconnect to clean up any DB2 CLI resources.

This attribute is used mainly by the Microsoft ODBC Driver Manager 3.5x before pooling the connection.

SQL_ATTR_CONNECTION_TIMEOUT
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_CONNECTTYPE
A 32-bit integer value that specifies whether this application is to operate in a coordinated or uncoordinated distributed environment. The possible values are:

- **SQL_CONCURRENT_TRANS (default)**: The application can have concurrent multiple connections to any one database or to multiple databases. Each connection has its own commit scope. No effort is made to enforce coordination of transactions. If an application issues a commit using the environment handle on `SQLEndTran()` and not all of the connections commit successfully, the application is responsible for recovery.
- SQL_COORDINATED_TRANS: The application wishes to have commit and rollbacks coordinated among multiple database connections. This option setting corresponds to the specification of the Type 2 CONNECT in embedded SQL. In contrast to the SQL_CONCURRENT_TRANS setting described above, the application is permitted only one open connection per database.

  Note: This connection type results in the default for SQL_ATTR_AUTOCOMMIT connection option to be SQL_AUTOCOMMIT_OFF.

If changing this attribute from the default then it must be set before any connections have been established on the environment handle.

It is recommended that the application set this attribute as an environment attribute with a call to SQLSetEnvAttr(), if necessary, as soon as the environment handle has been allocated. However, since ODBC applications cannot access SQLSetEnvAttr(), they must set this attribute using SQLSetConnectAttr() after each connection handle is allocated, but before any connections have been established.

All connections on an environment handle must have the same SQL_ATTR_CONNECTTYPE setting. An environment cannot have a mixture of concurrent and coordinated connections. The type of the first connection will determine the type of all subsequent connections. SQLSetEnvAttr() will return an error if an application attempts to change the connection type while there is an active connection.

The default connect type can also be set using the ConnectType CLI/ODBC configuration keyword.

Note: This is an IBM defined extension.

SQL_ATTR_CURRENT_CATALOG
A null-terminated character string containing the name of the catalog used by the data source. The catalog name is typically the same as the database name.

This connection attribute can be returned by SQLGetConnectAttr(), but cannot be set by SQLSetConnectAttr(). Any attempt to set this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_CURRENT_IMPLICIT_XMLPARSE_OPTION
A null-terminated character string that is the string constant used to set the CURRENT IMPLICIT XMLPARSE OPTION special register. Setting this attribute causes the SET CURRENT IMPLICIT XMLPARSE OPTION SQL statement to be issued. If this attribute is set before a connection has been established, the SET CURRENT IMPLICIT XMLPARSE OPTION SQL statement will be issued when the connection is made.

SQL_ATTR_CURRENT_PACKAGE_PATH
A null-terminated character string of package qualifiers that the DB2 database server uses to try to resolve the package when multiple packages have been configured. Setting this attribute causes the "SET CURRENT PACKAGE PATH = *schema1, schema2, ...*" statement to be issued after every connection to the database server.

This attribute is best suited for use with ODBC static processing applications, rather than CLI applications.

Note: This is an IBM defined extension.

SQL_ATTR_CURRENT_PACKAGE_SET

A null-terminated character string that indicates the schema name (collection identifier) that is used to select the package for subsequent SQL statements. Setting this attribute causes the SET CURRENT PACKAGESET SQL statement to be issued. If this attribute is set before a connection, the SET CURRENT PACKAGESET SQL statement will be issued at connection time.

CLI/ODBC applications issue dynamic SQL statements. Using this connection attribute, you can control the privileges used to run these statements:

- Choose a schema to use when running SQL statements from CLI/ODBC applications.
- Ensure the objects in the schema have the desired privileges and then rebind accordingly. This typically means binding the CLI packages (sqllib/bnd/db2cli.lst) using the COLLECTION <collid> option. Refer to the BIND command for further details.
- Set the CURRENTPACKAGESET option to this schema.

The SQL statements from the CLI/ODBC applications will now run under the specified schema and use the privileges defined there.

Setting the CurrentPackageSet CLI/ODBC configuration keyword is an alternative method of specifying the schema name.

The following package set names are reserved: NULLID, NULLIDR1, NULLIDRA.

SQL_ATTR_REOPT and SQL_ATTR_CURRENT_PACKAGE_SET are mutually exclusive, therefore, if one is set, the other is not allowed.

SQL_ATTR_CURRENT_SCHEMA
A null-terminated character string containing the name of the schema to be used by DB2 CLI for the `SQLColumns()` call if the *szSchemaName* pointer is set to null.

To reset this option, specify this option with a zero length string or a null pointer for the *ValuePtr* argument.

This option is useful when the application developer has coded a generic call to `SQLColumns()` that does not restrict the result set by schema name, but needs to constrain the result set at isolated places in the code.

This option can be set at any time and will be effective on the next `SQLColumns()` call where the *szSchemaName* pointer is null.

Note: This is an IBM defined extension.

SQL_ATTR_DB2_APPLICATION_HANDLE
A user-defined character string that returns the application handle of the connection. If the string is not large enough to contain the complete application handle, it will be truncated.

This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`.

SQL_ATTR_DB2_APPLICATION_ID
A user-defined character string that returns the application identifier of the connection. If the string is not large enough to contain the complete application identifier, it will be truncated.

This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`.

SQL_ATTR_DB2_SQLERRP
An sqlpointer to a null-terminated string containing the *sqlerrp* field of the sqlca.

Begins with a three-letter identifier indicating the product, followed by five digits indicating the version, release, and modification level of the product. For example, SQL08010 means DB2 UDB Version 8 Release 1 Modification level 0.

If SQLCODE indicates an error condition, then this field identifies the module that returned the error.

This field is also used when a successful connection is completed.

Note: This is an IBM defined extension.

SQL_ATTR_DB2ESTIMATE
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_DB2EXPLAIN
A 32-bit integer that specifies whether Explain snapshot, Explain mode information, or both should be generated by the server. Permitted values are:

- SQL_DB2EXPLAIN_OFF: Both the Explain Snapshot and the Explain table option facilities are disabled (a SET CURRENT EXPLAIN SNAPSHOT=NO and a SET CURRENT EXPLAIN MODE=NO are sent to the server).
- SQL_DB2EXPLAIN_SNAPSHOT_ON: The Explain Snapshot facility is enabled, and the Explain table option facility is disabled (a SET CURRENT EXPLAIN SNAPSHOT=YES and a SET CURRENT EXPLAIN MODE=NO are sent to the server).
- SQL_DB2EXPLAIN_MODE_ON: The Explain Snapshot facility is disabled, and the Explain table option facility is enabled (a SET CURRENT EXPLAIN SNAPSHOT=NO and a SET CURRENT EXPLAIN MODE=YES are sent to the server).
- SQL_DB2EXPLAIN_SNAPSHOT_MODE_ON: Both the Explain Snapshot and the Explain table option facilities are enabled (a SET CURRENT EXPLAIN SNAPSHOT=YES and a SET CURRENT EXPLAIN MODE=YES are sent to the server).

Before the explain information can be generated, the explain tables must be created.

This statement is not under transaction control and is not affected by a ROLLBACK. The new SQL_ATTR_DB2EXPLAIN setting is effective on the next statement preparation for this connection.

The current authorization ID must have INSERT privilege for the Explain tables.

The default value can also be set using the DB2Explain CLI/ODBC configuration keyword.

Note: This is an IBM defined extension.

SQL_ATTR_DECFLOAT_ROUNDING_MODE
A 32-bit integer that determines what decimal float rounding mode will be used for this connection. This attribute affects both the client and the server but only for actions initiated as part of this connection.

For a description of each of the rounding modes see the DecimalFloatRoundingMode CLI/ODBC configuration keyword

The options are:

- 0 = Half even (Default)
- 1 = Half up
- 2 = Down
- 3 = Ceiling

- 4 = Floor
- 5 = Half down
- 6 = Up

SQL_ATTR_DESCRIBE_CALL

A 32-bit integer value that indicates when stored procedure arguments are described. By default, DB2 CLI does not request input parameter describe information when it prepares a CALL statement. If an application has correctly bound parameters to a statement, then this describe information is unnecessary and not requesting it improves performance. The option values are:

- 1 = SQL_DESCRIBE_CALL_BEFORE.
- -1 = SQL_DESCRIBE_CALL_DEFAULT.

Setting this attribute can be done using the DescribeCall CLI/ODBC configuration keyword. Refer to the keyword for usage information and descriptions of the available options.

Note: This is an IBM defined extension.

SQL_ATTR_DESCRIBE_OUTPUT_LEVEL

A null-terminated character string that controls the amount of information the CLI driver requests on a prepare or describe request. By default, when the server receives a describe request, it returns the information contained in level 2 of Table 152 on page 336 for the result set columns. An application, however, might not need all of this information or might need additional information. Setting the SQL_ATTR_DESCRIBE_OUTPUT_LEVEL attribute to a level that suits the needs of the client application might improve performance because the describe data transferred between the client and server is limited to the minimum amount that the application requires. If the SQL_ATTR_DESCRIBE_OUTPUT_LEVEL setting is set too low, it might impact the functionality of the application (depending on the application's requirements). The DB2 CLI functions to retrieve the describe information might not fail in this case, but the information returned might be incomplete. Supported settings for SQL_ATTR_DESCRIBE_OUTPUT_LEVEL are:

- 0 - no describe information is returned to the client application
- 1 - describe information categorized in level 1 (see Table 152 on page 336) is returned to the client application
- 2 - (default) describe information categorized in level 2 (see Table 152 on page 336) is returned to the client application
- 3 - describe information categorized in level 3 (see Table 152 on page 336) is returned to the client application

The following table lists the fields that form the describe information that the server returns when it receives a prepare or describe request. These fields are grouped into levels, and the SQL_ATTR_DESCRIBE_OUTPUT_LEVEL attribute controls which levels of describe information the CLI driver requests.

Notes:

1. Not all levels of describe information are supported by all DB2 servers. All levels of describe information are supported on the following DB2 servers: DB2 on Linux, UNIX, and Windows Version 8 and later, DB2 for z/OS Version 8 and later, and DB2 UDB for iSeries Version 5

Release 3 and later. All other DB2 servers support only the 2 or 0 setting for SQL_ATTR_DESCRIBE_OUTPUT_LEVEL.

2. The default behavior will allow DB2 CLI to promote the level to 3 if the application asks for describe information that was not initially retrieved using the default level 2. This might result in two network flows to the server. If an application uses this attribute to explicitly set a describe level, then no promotion will occur. Therefore, if the attribute is used to set the describe level to 2, then DB2 CLI will not promote to level 3 even if the application asks for extended information.

Table 152. Levels of describe information

Level 1	Level 2	Level 3
SQL_DESC_COUNT SQL_COLUMN_COUNT SQL_DESC_TYPE SQL_DESC_CONCISE_TYPE SQL_COLUMN_LENGTH SQL_DESC_OCTET_LENGTH SQL_DESC_LENGTH SQL_DESC_PRECISION SQL_COLUMN_PRECISION SQL_DESC_SCALE SQL_COLUMN_SCALE SQL_DESC_DISPLAY_SIZE SQL_DESC_NULLABLE SQL_COLUMN_NULLABLE SQL_DESC_UNSIGNED SQL_DESC_SEARCHABLE SQL_DESC_LITERAL_SUFFIX SQL_DESC_LITERAL_PREFIX SQL_DESC_CASE_SENSITIVE SQL_DESC_FIXED_PREC_SCALE	all fields of level 1 and: SQL_DESC_NAME SQL_DESC_LABEL SQL_COLUMN_NAME SQL_DESC_UNNAMED SQL_DESC_TYPE_NAME SQL_DESC_DISTINCT_TYPE SQL_DESC_REFERENCE_TYPE SQL_DESC_STRUCTURED_TYPE SQL_DESC_USER_TYPE SQL_DESC_LOCAL_TYPE_NAME SQL_DESC_USER_DEFINED_ TYPE_CODE	all fields of levels 1 and 2 and: SQL_DESC_BASE_COLUMN_NAME SQL_DESC_UPDATABLE SQL_DESC_AUTO_UNIQUE_VALUE SQL_DESC_SCHEMA_NAME SQL_DESC_CATALOG_NAME SQL_DESC_TABLE_NAME SQL_DESC_BASE_TABLE_NAME

Setting the DescribeOutputLevel CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_ENLIST_IN_DTC

An SQLPOINTER which can be either of the following:

- non-null transaction pointer: The application is asking the DB2 CLI/ODBC driver to change the state of the connection from non-distributed transaction state to distributed state. The connection will be enlisted with the Distributed Transaction Coordinator (DTC).
- null: The application is asking the DB2 CLI/ODBC driver to change the state of the connection from distributed transaction state to a non-distributed transaction state.

This attribute is only used in a Microsoft Transaction Server (MTS) environment to enlist or un-enlist a connection with MTS.

Each time this attribute is used with a non-null transaction pointer, the previous transaction is assumed to be ended and a new transaction is initiated. The application must call the ITransaction member function Endtransaction before calling this API with a non-null pointer. Otherwise the previous transaction will be aborted. The application can enlist multiple connections with the same transaction pointer.

Note: This connection attribute is specified by MTS automatically for each transaction and is not coded by the user application.

It is imperative for CLI/ODBC applications that there will be no

concurrent SQL statements executing on 2 different connections into the same database that are enlisted in the same transaction.

SQL_ATTR_FREE_LOCATORS_ON_FETCH
A boolean attribute that specifies if LOB locators are freed when `SQLFetch()` is executed, rather than when a COMMIT is issued. Setting this attribute to 1 (true) frees the locators that are used internally when applications fetch LOB data without binding the LOB columns with `SQLBindCol()` (or equivalent descriptor APIs). Locators that are explicitly returned to the application must still be freed by the application. This attribute value can be used to avoid scenarios where an application receives SQLCODE = -429 (no more locators). The default for this attribute is 0 (false).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_ACCTSTR
A pointer to a null-terminated character string used to identify the client accounting string sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 200 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.)

Note: This is an IBM defined extension.

SQL_ATTR_INFO_APPLNAME
A pointer to a null-terminated character string used to identify the client application name sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 32 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_PROGRAMID
A user-defined character string, with a maximum length of 80 bytes, that associates an application with a connection. Once this attribute is set, DB2 UDB for z/OS Version 8 associates this identifier with any statements inserted into the dynamic SQL statement cache.

This attribute is only supported for CLI applications accessing DB2 UDB for z/OS Version 8.

Note: This is an IBM defined extension.

SQL_ATTR_INFO_PROGRAMNAME
A null-terminated user-defined character string, up to 20 bytes in length, used to specify the name of the application running on the client.

When this attribute is set before the connection to the server is established, the value specified overrides the actual client application name and will be the value that is displayed in the appl_name monitor element. When connecting to a DB2 UDB for z/OS server, the first 12 characters of this setting are used as the CORRELATION IDENTIFIER of the associated DB2 UDB for z/OS thread.

Note: This is an IBM defined extension.

SQL_ATTR_INFO_USERID
A pointer to a null-terminated character string used to identify the client user ID sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 16 characters.
- This user-id is not to be confused with the authentication user-id. This user-id is for identification purposes only and is not used for any authorization.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_INFO_WRKSTNNAME
A pointer to a null-terminated character string used to identify the client workstation name sent to the host database server when using DB2 Connect.

Please note:

- When the value is being set, some servers might not handle the entire length provided and might truncate the value.
- DB2 for z/OS and OS/390 servers support up to a length of 18 characters.
- To ensure that the data is converted correctly when transmitted to a host system, use only the characters A to Z, 0 to 9, and the underscore (_) or period (.).

Note: This is an IBM defined extension.

SQL_ATTR_KEEP_DYNAMIC
A 32-bit unsigned integer value which specifies whether the KEEPDYNAMIC option has been enabled. If enabled, the server will keep dynamically prepared statements in a prepared state across transaction boundaries.

- 0 - KEEPDYNAMIC functionality is not available; CLI packages were bound with the KEEPDYNAMIC NO option
- 1 - KEEPDYNAMIC functionality is available; CLI packages were bound with the KEEPDYNAMIC YES option

It is recommended that when this attribute is set, the SQL_ATTR_CURRENT_PACKAGE_SET attribute also be set.

Note: This is an IBM defined extension.

SQL_ATTR_LOB_CACHE_SIZE
A 32-bit unsigned integer that specifies maximum cache size (in bytes) for LOBs. By default, LOBs are not cached.

See the LOBCacheSize CLI/ODBC configuration keyword for further usage information.

SQL_ATTR_LOGIN_TIMEOUT
A 32-bit integer value corresponding to the number of seconds to wait for a reply when trying to establish a connection to a server before terminating the attempt and generating a communication timeout. Specify a positive integer, up to 32 767. The default setting of 0 will allow the client to wait indefinitely.

Setting a connection timeout value can also be done using the ConnectTimeout CLI/ODBC configuration keyword. Refer to the keyword for usage information.

SQL_ATTR_LONGDATA_COMPAT
A 32-bit integer value indicating whether the character, double byte character and binary large object data types should be reported respectively as SQL_LONGVARCHAR, SQL_LONGVARGRAPHIC or SQL_LONGBINARY, enabling existing applications to access large object data types seamlessly. The option values are:

- **SQL_LD_COMPAT_NO (default)**: The large object data types are reported as their respective IBM-defined types (SQL_BLOB, SQL_CLOB, SQL_DBCLOB).
- SQL_LD_COMPAT_YES: The IBM large object data types (SQL_BLOB, SQL_CLOB and SQL_DBCLOB) are mapped to SQL_LONGVARBINARY, SQL_LONGVARCHAR and SQL_LONGVARGRAPHIC; `SQLGetTypeInfo()` returns one entry each for SQL_LONGVARBINARY SQL_LONGVARCHAR, and SQL_LONGVARGRAPHIC.

Note: This is an IBM defined extension.

SQL_ATTR_MAPCHAR
A 32-bit integer value used to specify the default SQL type associated with SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR. The option values are:

- **SQL_MAPCHAR_DEFAULT (default)**: return the default SQL type representation
- SQL_MAPCHAR_WCHAR: return SQL_CHAR as SQL_WCHAR, SQL_VARCHAR as SQL_WVARCHAR, and SQL_LONGVARCHAR as SQL_WLONGVARCHAR

Only the following DB2 CLI functions are affected by setting this attribute:

- SQLColumns()
- SQLColAttribute()
- SQLDescribeCol()
- SQLDescribeParam()
- SQLGetDescField()
- SQLGetDescRec()

- SQLProcedureColumns()

Setting the default SQL type associated with SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR can also be done using the MapCharToWChar CLI/ODBC configuration keyword.

Note: This is an IBM defined extension.

SQL_ATTR_MAXCONN
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_MAX_LOB_BLOCK_SIZE
A 32-bit unsigned integer that indicates the maximum size of LOB or XML data block. Specify a positive integer, up to 2 147 483 647. The default setting of 0 indicates that there is no limit to the data block size for LOB or XML data.

During data retrieval, the server will include all of the information for the current row in its reply to the client even if the maximum block size has been reached.

If both MaxLOBBlockSize and the db2set registry variable DB2_MAX_LOB_BLOCK_SIZE are specified, the value for MaxLOBBlockSize will be used.

Setting the MaxLOBBlockSize CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_METADATA_ID
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_ODBC_CURSORS
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_PACKET_SIZE
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_PING_DB
A 32-bit integer which is used with `SQLGetConnectAttr()` to get the ping time in microseconds.

If a connection has previously been established and has been dropped by the database, a value of 0 is reported. If the connection has been closed by the application, then an SQLSTATE of 08003 is reported. This connection attribute can be returned by `SQLGetConnectAttr()`, but cannot be set by `SQLSetConnectAttr()`. Any attempt to set this attribute will result in an SQLSTATE of 7HYC00 (Driver not capable)

Note: This is an IBM defined extension.

SQL_ATTR_QUIET_MODE
A 32-bit platform specific window handle.

If the application has never made a call to `SQLSetConnectAttr()` with this option, then DB2 CLI would return a null parent window handle on `SQLGetConnectAttr()` for this option and use a null parent window handle

to display dialogue boxes. For example, if the end user has asked for (via an entry in the DB2 CLI initialization file) optimizer information to be displayed, DB2 CLI would display the dialogue box containing this information using a null window handle. (For some platforms, this means the dialogue box would be centered in the middle of the screen.)

If *ValuePtr* is set to null , then DB2 CLI does not display any dialogue boxes. In the above example where the end user has asked for the optimizer estimates to be displayed, DB2 CLI would not display these estimates because the application explicitly wants to suppress all such dialogue boxes.

If *ValuePtr* is not null, then it should be the parent window handle of the application. DB2 CLI uses this handle to display dialogue boxes. (For some platforms, this means the dialogue box would be centered with respect to the active window of the application.)

Note: This connection option cannot be used to suppress the `SQLDriverConnect()` dialogue box (which can be suppressed by setting the *fDriverCompletion* argument to SQL_DRIVER_NOPROMPT).

SQL_ATTR_RECEIVE_TIMEOUT

A 32-bit integer value that is the number of seconds a client waits for a reply from a server on an established connection before terminating the attempt and generating a communication timeout error. The default value of 0 indicates the client waits indefinitely for a reply. The receive timeout has no effect during connection establishment; it is only supported for TCP/IP, and is ignored for any other protocol. Supported values are integers from 0 to 32767.

Note: This is an IBM defined extension.

SQL_ATTR_REOPT

A 32-bit integer value that enables query optimization for SQL statements that contain special registers or parameter markers. Optimization occurs by using the values available at query execution time for special registers or parameter markers, instead of the default estimates that are chosen by the compiler. The valid values of the attribute are:

- **2 = SQL_REOPT_NONE (default):** No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for the special registers or parameter markers. The default NULLID package set is used to execute dynamic SQL statements.
- 3 = SQL_REOPT_ONCE: Query optimization occurs once at query execution time, when the query is executed for the first time. The NULLIDR1 package set, which is bound with the REOPT ONCE bind option, is used.
- 4 = SQL_REOPT_ALWAYS: Query optimization or reoptimization occurs at query execution time every time the query is executed. The NULLIDRA package set, which is bound with the REOPT ALWAYS bind option, is used.

The NULLIDR1 and NULLIDRA are reserved package set names, and when used, REOPT ONCE and REOPT ALWAYS are implied respectively. These package sets have to be explicitly created with these commands:

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

SQL_ATTR_REOPT and SQL_ATTR_CURRENT_PACKAGE_SET are mutually exclusive, therefore, if one is set, the other is not allowed.

Note: This is an IBM defined extension.

SQL_ATTR_REPORT_ISLONG_FOR_LONGTYPES_OLEDB
A 32-bit integer value. The OLE DB client cursor engine and the OLE DB .NET Data Provider CommandBuilder object generate UPDATE and DELETE statements based on column information provided by the IBM DB2 OLE DB Provider. If the generated statement contains a LONG type in the WHERE clause, the statement will fail because LONG types cannot be used in a search with an equality operator. The possible values are:

- **0 (default)**: LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) do not have the DBCOLUMNFLAGS_ISLONG flag set, which might cause the columns to be used in the WHERE clause.
- 1: The IBM DB2 OLE DB Provider reports LONG types (LONG VARCHAR, LONG VARCHAR FOR BIT DATA, LONG VARGRAPHIC and LONG VARGRAPHIC FOR BIT DATA) with the DBCOLUMNFLAGS_ISLONG flag set. This will prevent the long columns from being used in the WHERE clause.

This attribute is supported by the following database servers:

- DB2 UDB for z/OS
  - version 6 with PTF UQ93891
  - version 7 with PTF UQ93889
  - version 8 with PTF UQ93890
  - versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
  - version 8.2 (equivalent to Version 8.1, FixPak 7) and later

Note: This is an IBM defined extension.

SQL_ATTR_SERVER_MSGTXT_MASK
A 32-bit integer value used to indicate when DB2 CLI should request the error message from the server. This attribute is used in conjunction with the SQL_ATTR_SERVER_MSGTXT_SP attribute. The attribute can be set to:

- **SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST (default)**: DB2 CLI will check the local message files first to see if the message can be retrieved. If no matching SQLCODE is found, then DB2 CLI will request the information from the server.
- SQL_ATTR_SERVER_MSGTXT_MASK_WARNINGS: DB2 CLI always requests the message information from the server for warnings but error messages are retrieved from the local message files.
- SQL_ATTR_SERVER_MSGTXT_MASK_ERRORS: DB2 CLI always requests the message information from the server for errors but warning messages are retrieved from the local message files.
- SQL_ATTR_SERVER_MSGTXT_MASK_ALL: DB2 CLI always requests the message information from the server for both error and warning messages.

Setting the ServerMsgMask CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

Note: This is an IBM defined extension.

SQL_ATTR_SERVER_MSGTXT_SP

A pointer to a character string used to identify a stored procedure that is used for generating an error message based on an SQLCA. This can be useful when retrieving error information from a server such as DB2 UDB z/OS. The attribute can be set to:

- DSNACCMG: The default procedure on DB2 UDB for z/OS that can be used to retrieve the message text from the server.

Applications using this attribute can also set the SQL_ATTR_SERVER_MSGTXT_MASK attribute to indicate when DB2 CLI should call this procedure to retrieve the message information from the server. If the SQL_ATTR_SERVER_MSGTXT_MASK is not set, then the default is to check the local message files first (see SQL_ATTR_SERVER_MSGTXT_MASK_LOCAL_FIRST in SQL_ATTR_SERVER_MSGTXT_MASK).

Setting the UseServerMsgSP CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

Note: This is an IBM defined extension.

SQL_ATTR_SQLCOLUMNS_SORT_BY_ORDINAL_OLEDB

A 32–bit integer value. The Microsoft OLE DB specification requires that IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMNS) returns the row set sorted by the columns TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME. The IBM DB2 OLE DB Provider conforms to the specification, however, applications that use the Microsoft ODBC Bridge provider (MSDASQL) have been typically coded to get the row set sorted by ORDINAL_POSITION. The possible values are:

- **0 (default)**: The IBM DB2 OLE DB Provider returns a row set sorted by the columns TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME.
- 1: The IBM DB2 OLE DB Provider returns a row set sorted by ORDINAL_POSITION.

This attribute is supported by the following database servers:

- DB2 UDB for z/OS
  - version 6 with PTF UQ93891
  - version 7 with PTF UQ93889
  - version 8 with PTF UQ93890
  - versions later than version 8, PTFs are not required
- DB2 Database for Linux, UNIX, and Windows
  - version 8.2 (equivalent to Version 8.1, FixPak 7) and later

Note: This is an IBM defined extension.

SQL_ATTR_STREAM_GETDATA

A 32-bit unsigned integer that indicates if the data output stream for the `SQLGetData()` function will be optimized. The values are:

- **0 (default)**: DB2 CLI buffers all the data on the client.
- 1: For applications that do not need to buffer data and are querying data on a server that supports Dynamic Data Format, specify 1 to indicate that data buffering is not required. The DB2 CLI client will optimize the data output stream.

This keyword is ignored if Dynamic Data Format is not supported by the server.

If StreamGetData is set to 1 and DB2 CLI cannot determine the number of bytes still available to return in the output buffer, `SQLGetData()` returns SQL_NO_TOTAL (-4) as the length when truncation occurs. Otherwise, `SQLGetData()` returns the number of bytes still available.

Setting the StreamGetData CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_SYNC_POINT
This attribute has been deprecated in DB2 UDB Version 8.

SQL_ATTR_TRACE
This connection attribute can be set by an application for the ODBC Driver Manager. Any attempt to set this connection attribute for the DB2 CLI Driver will result in an SQLSTATE of HYC00 (Driver not capable).

Instead of using this connection attribute, the DB2 CLI trace facility can be set using the Trace CLI/ODBC configuration keyword. Alternatively, the environment attribute SQL_ATTR_TRACE can be used to configure tracing features. Note that the environment attribute does not use the same syntax as the ODBC Driver Manager's connection attribute.

SQL_ATTR_TRACEFILE
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

Instead of using this attribute, the DB2 CLI trace file name is set using the TraceFileName CLI/ODBC configuration keyword.

SQL_ATTR_TRANSLATE_LIB
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute on other platforms will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_TRANSLATE_OPTION
This connection attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute on other platforms will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_TRUSTED_CONTEXT_PASSWORD
A user defined string containing a password. Use this attribute if the database server requires a password when switching users on a trusted connection. Set this attribute after setting the attribute SQL_ATTR_TRUSTED_CONTEXT_USERID and before executing any SQL statements that access the database server. If SQL_ATTR_TRUSTED_CONTEXT_USERID is not set before setting this attribute, an error (CLI0198E) is returned.

SQL_ATTR_TRUSTED_CONTEXT_USERID

A user defined string containing a user ID. Use this on existing trusted connections to switch users. Do not use it when creating a trusted connection.

After setting this attribute the user switch will occur the next time that you execute an SQL statement that accesses the database server. (SQLSetConnectAttr does not access the database server.) If the user switch is successful the user ID in this attribute becomes the new user of the

connection. If the user switch fails the call that initiated the switch will return an error indicating the reason for the failure.

The user ID must be a valid authorization ID on the database server unless you are using an identity server, in which case you can use any user name recognized by the identity server. (If you are using an identity server see also "SQL_ATTR_USER_REGISTRY_NAME" on page 346.)

If you set this attribute while the connection handle is not yet connected to a database or if the connection is not a trusted connection then an error (CLI0197E) is returned.

SQL_ATTR_TXN_ISOLATION

A 32-bit bitmask that sets the transaction isolation level for the current connection referenced by *ConnectionHandle*. The valid values for *ValuePtr* can be determined at runtime by calling `SQLGetInfo()` with *fInfoType* set to SQL_TXN_ISOLATION_OPTIONS. The following values are accepted by DB2 CLI, but each server might only support a subset of these isolation levels:

- SQL_TXN_READ_UNCOMMITTED - Dirty reads, non-repeatable reads, and phantom reads are possible.
- **SQL_TXN_READ_COMMITTED (default)** - Dirty reads are not possible. Non-repeatable reads and phantom reads are possible.
- SQL_TXN_REPEATABLE_READ - Dirty reads and reads that cannot be repeated are not possible. Phantoms are possible.
- SQL_TXN_SERIALIZABLE - Transactions can be serialized. Dirty reads, non-repeatable reads, and phantoms are not possible.
- SQL_TXN_NOCOMMIT - Any changes are effectively committed at the end of a successful operation; no explicit commit or rollback is allowed. This is analogous to autocommit. This is not an SQL92 isolation level, but an IBM defined extension, supported only by DB2 UDB for AS/400.

In IBM terminology,

- SQL_TXN_READ_UNCOMMITTED is Uncommitted Read;
- SQL_TXN_READ_COMMITTED is Cursor Stability;
- SQL_TXN_REPEATABLE_READ is Read Stability;
- SQL_TXN_SERIALIZABLE is Repeatable Read.

This option cannot be specified while there is an open cursor on any statement handle, or an outstanding transaction for this connection; otherwise, SQL_ERROR is returned on the function call (SQLSTATE **S1**011).

This attribute (or corresponding keyword) is only applicable if the default isolation level is used. If the application has specifically set the isolation level then this attribute will have no effect.

Note: There is an IBM extension that permits the setting of transaction isolation levels on a per statement handle basis. See the SQL_ATTR_STMTTXN_ISOLATION statement attribute.

SQL_ATTR_USE_TRUSTED_CONTEXT

When connecting to a DB2 database server that supports trusted contexts, set this attribute if you want the connection you are creating to be a trusted connection. If this attribute is set to SQL_TRUE and the database server determines that the connection can be trusted then the connection is a trusted connection. If this attribute is not set, if it is set to SQL_FALSE, if the database server does not support trusted contexts, or if the database server determines that the connection cannot be trusted then a regular

connection is created instead and a warning (SQLSTATE 01679) is returned. This value can only be specified before the connection is established either for the first time or following a call to the `SQLDisconnect()` function.

SQL_ATTR_USER_REGISTRY_NAME

This attribute is only used when authenticating a user on a server that is using an identity mapping service. It is set to a user defined string that names an identity mapping registry. The format of the registry name varies depending on the identity mapping service used. By providing this attribute you tell the server that the user name provided can be found in this registry.

After setting this attribute the value will be used on subsequent attempts to establish a normal connection, establish a trusted connection, or switch the user id on a trusted connection.

SQL_ATTR_WCHARTYPE

A 32-bit integer that specifies, in a double-byte environment, which wchar_t (SQLDBCHAR) character format you want to use in your application. This option provides you the flexibility to choose between having your wchar_t data in multi-byte format or in wide-character format. There two possible values for this option:

- SQL_WCHARTYPE_CONVERT: character codes are converted between the graphic SQL data in the database and the application variable. This allows your application to fully exploit the ANSI C mechanisms for dealing with wide character strings (for example, L-literals, 'wc' string functions) without having to explicitly convert the data to multi-byte format before communicating with the database. The disadvantage is that the implicit conversions might have an impact on the runtime performance of your application, and might increase memory requirements. If you want WCHARTYPE CONVERT behavior then define the C preprocessor macro SQL_WCHART_CONVERT at compile time. This ensures that certain definitions in the DB2 header files use the data type wchar_t instead of sqldbchar.
- **SQL_WCHARTYPE_NOCONVERT (default)**: no implicit character code conversion occurs between the application and the database. Data in the application variable is sent to and received from the database as unaltered DBCS characters. This allows the application to have improved performance, but the disadvantage is that the application must either refrain from using wide-character data in wchar_t (SQLDBCHAR) application variables, or it must explicitly call the wcstombs() and mbstowcs() ANSI C functions to convert the data to and from multi-byte format when exchanging data with the database.

Note: This is an IBM defined extension.

SQL_ATTR_XML_DECLARATION

A 32-bit unsigned integer that specifies which elements of an XML declaration are added to XML data when it is implicitly serialized. This attribute does not affect the result of the XMLSERIALIZE function. Set this attribute to the sum of each component required:

- 0: No declarations or byte order marks (BOMs) are added to the output buffer.

- 1: A byte order mark (BOM) in the appropriate endianness is prepended to the output buffer if the target encoding is UTF-16 or UTF-32. (Although a UTF-8 BOM exists, DB2 does not generate it, even if the target encoding is UTF-8.)
- 2: A minimal XML declaration is generated, containing only the XML version.
- 4: An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this attribute.

Attempts to set any other value using `SQLSetConnectAttr()` or `SQLSetConnectOption()` will result in a CLI0191E (SQLSTATE HY024) error, and the value will remain unchanged.

The default setting is 7, which indicates that a BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.

This setting affects any statement handles allocated after the value is changed. Existing statement handles retain their original values.

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Multithreaded CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Isolation levels" in *SQL Reference, Volume 1*
- "Unicode CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Unicode functions (CLI)" in *Call Level Interface Guide and Reference, Volume 1*
- "XML data handling in CLI applications - Overview" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*
- "DecimalFloatRoundingMode CLI/ODBC configuration keyword" in *Call Level Interface Guide and Reference, Volume 1*
- "CURRENT IMPLICIT XMLPARSE OPTION special register" in *SQL Reference, Volume 1*
- "sqleBeginCtx API - Create and attach to an application context" in *Administrative API Reference*
- "DBCS character sets" in *Developing SQL and External Routines*
- "SQLGetConnectAttr function (CLI) - Get current attribute setting" on page 146
- "SQLGetStmtAttr function (CLI) - Get current setting of a statement attribute" on page 216
- "SQLSetConnectAttr function (CLI) - Set connection attributes" on page 266
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "SQLSetStmtOption function (CLI) - Set statement option" on page 299
- "SET CURRENT IMPLICIT XMLPARSE OPTION statement" in *SQL Reference, Volume 2*
- "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*

- "XMLSERIALIZE scalar function" in *SQL Reference, Volume 1*
- "BIND command" in *Command Reference*

Statement attributes (CLI) list

The currently defined attributes and the version of DB2 CLI or ODBC in which they were introduced are shown below; it is expected that more will be defined to take advantage of different data sources.

SQL_ATTR_APP_PARAM_DESC
The handle to the APD for subsequent calls to `SQLExecute()` and `SQLExecDirect()` on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement was initially allocated. If this attribute is set to SQL_NULL_DESC, an explicitly allocated APD handle that was previously associated with the statement handle is dissociated from it, and the statement handle reverts to the implicitly allocated APD handle.

This attribute cannot be set to a descriptor handle that was implicitly allocated for another statement or to another descriptor handle that was implicitly set on the same statement; implicitly allocated descriptor handles cannot be associated with more than one statement or descriptor handle.

This attribute cannot be set at the connection level.

SQL_ATTR_APP_ROW_DESC
The handle to the ARD for subsequent fetches on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement was initially allocated. If this attribute is set to SQL_NULL_DESC, an explicitly allocated ARD handle that was previously associated with the statement handle is dissociated from it, and the statement handle reverts to the implicitly allocated ARD handle.

This attribute cannot be set to a descriptor handle that was implicitly allocated for another statement or to another descriptor handle that was implicitly set on the same statement; implicitly allocated descriptor handles cannot be associated with more than one statement or descriptor handle.

This attribute cannot be set at the connection level.

SQL_ATTR_APP_USES_LOB_LOCATOR
A 32-bit unsigned integer that indicates if applications are using LOB locators. This attribute has either of the following values:

- **1 (default)**: Indicates that applications are using LOB locators.
- 0: For applications that do not use LOB locators and are querying data on a server that supports Dynamic Data Format, specify 0 to indicate that LOB locators are not used and allow the return of LOB data to be optimized.

This keyword is ignored for stored procedure result sets.

If the keyword is set to 0 and an application binds a LOB locator to a result set using `SQLBindCol()`, an Invalid conversion error will be returned by the `SQLFetch()` function.

Setting the AppUsesLOBLocator CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_ASYNC_ENABLE

A 32-bit integer value that specifies whether a function called with the specified statement is executed asynchronously:

- **SQL_ASYNC_ENABLE_OFF** = Off (the default)
- SQL_ASYNC_ENABLE_ON = On

Once a function has been called asynchronously, only the original function, `SQLAllocHandle()`, `SQLCancel()`, `SQLSetStmtAttr()`, `SQLGetDiagField()`, `SQLGetDiagRec()`, or `SQLGetFunctions()` can be called on the statement handle, until the original function returns a code other than SQL_STILL_EXECUTING. Any other function called on any other statement handle under the same connection returns SQL_ERROR with an SQLSTATE of HY010 (Function sequence error).

Because DB2 CLI supports statement level asynchronous-execution, the statement attribute SQL_ATTR_ASYNC_ENABLE can be set. Its initial value is the same as the value of the connection level attribute with the same name at the time the statement handle was allocated.

The following functions can be executed asynchronously: `SQLBulkOperations()`, `SQLColAttribute()`, `SQLColumnPrivileges()`, `SQLColumns()`, `SQLDescribeCol()`, `SQLDescribeParam()`, `SQLExecDirect()`, `SQLExecute()`, `SQLExtendedFetch()`, `SQLExtendedPrepare()`, `SQLFetch()`, `SQLFetchScroll()`, `SQLForeignKeys()`, `SQLGetData()`, `SQLGetLength()`, `SQLGetPosition()`, `SQLMoreResults()`, `SQLNumResultCols()`, `SQLParamData()`, `SQLPrepare()`, `SQLPrimaryKeys()`, `SQLProcedureColumns()`, `SQLProcedures()`, `SQLRowCount()`, `SQLSetPos()`, `SQLSpecialColumns()`, `SQLStatistics()`, `SQLTablePrivileges()`, `SQLTables()`.

Note: Any Unicode equivalent of a function stated above can be called asynchronously.

SQL_ATTR_BLOCK_FOR_NROWS

A 32-bit integer that specifies the desired block size, in rows, to be returned by the server when fetching a result set. For large read-only result sets consisting of one or more data blocks, a large block size can improve performance by reducing the number of synchronous server block requests made by the client. The default value is 0 which means the default block size will be returned by the server.

SQL_ATTR_BLOCK_LOBS

A Boolean attribute that specifies if LOB blocking fetch is enabled. By default, this attribute is set to 0 (false), however, when set to 1 (true) and when accessing a server that supports LOB blocking, all of the LOB data associated with rows that fit completely within a single query block are returned in a single fetch request.

SQL_ATTR_CALL_RETURN

A read-only attribute to be retrieved after executing a stored procedure. The value returned from this attribute is -1 if the stored procedure failed to execute (for example, if the library containing the stored procedure executable cannot be found). If the stored procedure executed successfully but has a negative return code (for example, if data truncation occurred when inserting data into a table), then SQL_ATTR_CALL_RETURN will return the value that was set in the sqlerrd(1) field of the SQLCA when the stored procedure was executed.

SQL_ATTR_CHAINING_BEGIN

A 32-bit integer which specifies that DB2 will chain together `SQLExecute()` requests for a single prepared statement before sending the requests to the server; this feature is referred to as CLI array input chaining. All `SQLExecute()` requests associated with a prepared statement will not be sent to the server until either the SQL_ATTR_CHAINING_END statement attribute is set, or the available buffer space is consumed by rows that have been chained. The size of this buffer is defined by the ASLHEAPSZ database manager configuration parameter for local client applications, or the RQRIOBLK database manager configuration parameter for client/server configurations.

This attribute can be used with the CLI/ODBC configuration keyword ArrayInputChain to effect array input without needing to specify the array size. Refer to the documentation for ArrayInputChain for more information.

Note: The specific 32-bit integer value that is set with this attribute is not significant to DB2 CLI. Simply setting this attribute to any 32-bit integer value will enable the CLI array input chaining feature.

SQL_ATTR_CHAINING_END

A 32-bit integer which specifies that the CLI array input chaining behavior enabled earlier, with the setting of the SQL_ATTR_CHAINING_BEGIN statement attribute, ends. Setting SQL_ATTR_CHAINING_END causes all chained `SQLExecute()` requests to be sent to the server. After this attribute is set, `SQLRowCount()` can be called to determine the total row count for all `SQLExecute()` statements that were chained between the SQL_ATTR_CHAINING_BEGIN and SQL_ATTR_CHAINING_END pair. Error diagnostic information for the chained statements becomes available after the SQL_ATTR_CHAINING_END attribute is set.

This attribute can be used with the DB2 CLI configuration keyword ArrayInputChain to effect array input without needing to specify the array size. Refer to the documentation for ArrayInputChain for more information.

Note: The specific 32-bit integer value that is set with this attribute is not significant to DB2 CLI. Simply setting this attribute to any 32-bit integer value will disable the CLI array input chaining feature that was enabled when SQL_ATTR_CHAINING_BEGIN was set.

SQL_ATTR_CLIENT_LOB_BUFFERING

Specifies whether LOB locators or the underlying LOB data is returned in a result set for LOB columns that are not bound. By default, locators are returned. If an application usually fetches unbound LOBs and then must retrieve the underlying LOB data, the application's performance can be improved by retrieving the LOB data from the outset; this reduces the number of synchronous waits and network flows. The possible values for this attribute are:

- SQL_CLIENTLOB_USE_LOCATORS (default) - LOB locators are returned
- SQL_CLIENTLOB_BUFFER_UNBOUND_LOBS - actual LOB data is returned

SQL_ATTR_CLOSE_BEHAVIOR

A 32-bit integer that specifies whether the DB2 server should attempt to release read locks acquired during a cursor's operation when the cursor is closed. It can be set to either:

- **SQL_CC_NO_RELEASE** - read locks are not released. This is the default.
- SQL_CC_RELEASE - read locks are released.

For cursors opened with isolation UR or CS, read locks are not held after a cursor moves off a row. For cursors opened with isolation RS or RR, SQL_ATTR_CLOSE_BEHAVIOR modifies some of those isolation levels, and an RR cursor might experience nonrepeatable reads or phantom reads.

If a cursor that is originally RR or RS is reopened after being closed with SQL_ATTR_CLOSE_BEHAVIOR then new read locks will be acquired.

This attribute can also be set at the connection level, however when set at the connection level, it only affects cursor behavior for statement handles that are opened after this attribute is set.

Refer to the `SQLCloseCursor()` function for more information.

SQL_ATTR_CLOSEOPEN

To reduce the time it takes to open and close cursors, DB2 will automatically close an open cursor if a second cursor is opened using the same handle. Network flow is therefore reduced when the close request is chained with the open request and the two statements are combined into one network request (instead of two).

- **0** = DB2 acts as a regular ODBC data source: Do not chain the close and open statements, return an error if there is an open cursor. This is the default.
- 1 = Chain the close and open statements.

Previous CLI applications will not benefit from this default because they are designed to explicitly close the cursor. New applications, however, can take advantage of this behavior by not closing the cursors explicitly, but by allowing CLI to close the cursor on subsequent open requests.

SQL_ATTR_CONCURRENCY

A 32-bit integer value that specifies the cursor concurrency:

- SQL_CONCUR_READ_ONLY = Cursor is read-only. No updates are allowed. Supported by forward-only, static and keyset cursors.
- SQL_CONCUR_LOCK = Cursor uses the lowest level of locking sufficient to ensure that the row can be updated. Supported by forward-only and keyset cursors.
- SQL_CONCUR_VALUES = Cursor uses optimistic concurrency control, comparing values.

The default value for SQL_ATTR_CONCURRENCY is SQL_CONCUR_READ_ONLY for static and forward-only cursors. The default for a keyset cursor is SQL_CONCUR_VALUES.

This attribute cannot be specified for an open cursor.

If the SQL_ATTR_CURSOR_TYPE *Attribute* is changed to a type that does not support the current value of SQL_ATTR_CONCURRENCY, the value of SQL_ATTR_CONCURRENCY will be changed at execution time, and a warning issued when `SQLExecDirect()` or `SQLPrepare()` is called.

If a SELECT FOR UPDATE statement is executed while the value of SQL_ATTR_CONCURRENCY is set to SQL_CONCUR_READ_ONLY, an

error will be returned. If the value of SQL_ATTR_CONCURRENCY is changed to a value that is supported for some value of SQL_ATTR_CURSOR_TYPE, but not for the current value of SQL_ATTR_CURSOR_TYPE, the value of SQL_ATTR_CURSOR_TYPE will be changed at execution time, and SQLSTATE 01S02 (Option value changed) is issued when `SQLExecDirect()` or `SQLPrepare()` is called.

If the specified concurrency is not supported by the data source, then DB2 CLI substitutes a different concurrency and returns SQLSTATE 01S02 (Option value changed). The order of substitution depends on the cursor type:

- Forward-Only: SQL_CONCUR_LOCK is substituted for SQL_CONCUR_ROWVER and SQL_CONCUR_VALUES
- Static: only SQL_CONCUR_READ_ONLY is valid
- Keyset: SQL_CONCUR_VALUES is substituted for SQL_CONCUR_ROWVER

Note: The following value has also been defined by ODBC, but is not supported by DB2 CLI
- SQL_CONCUR_ROWVER = Cursor uses optimistic concurrency control.

SQL_ATTR_CURSOR_HOLD

A 32-bit integer which specifies whether the cursor associated with this *StatementHandle* is preserved in the same position as before the COMMIT operation, and whether the application can fetch without executing the statement again.
- **SQL_CURSOR_HOLD_ON** (this is the default)
- SQL_CURSOR_HOLD_OFF

The default value when an *StatementHandle* is first allocated is SQL_CURSOR_HOLD_ON.

This option cannot be specified while there is an open cursor on this *StatementHandle*.

The default cursor hold mode can also be set using the CURSORHOLD DB2 CLI/ODBC configuration keyword.

Note: This option is an IBM extension.

SQL_ATTR_CURSOR_SCROLLABLE

A 32-bit integer that specifies the level of support that the application requires. Setting this attribute affects subsequent calls to `SQLExecute()` and `SQLExecDirect()`. The supported values are:
- **SQL_NONSCROLLABLE** = Scrollable cursors are not required on the statement handle. If the application calls `SQLFetchScroll()` on this handle, the only valid value of *FetchOrientation()* is SQL_FETCH_NEXT. This is the default.
- SQL_SCROLLABLE = Scrollable cursors are required on the statement handle. When calling `SQLFetchScroll()`, the application can specify any valid value of *FetchOrientation*, achieving cursor positioning in modes other than the sequential mode.

SQL_ATTR_CURSOR_SENSITIVITY

A 32-bit integer that specifies whether cursors on the statement handle make visible the changes made to a result set by another cursor. Setting

this attribute affects subsequent calls to `SQLExecute()` and `SQLExecDirect()`. The supported values are:

- **SQL_UNSPECIFIED** = It is unspecified what the cursor type is and whether cursors on the statement handle make visible the changes made to a result set by another cursor. Cursors on the statement handle might make visible none, some or all such changes. This is the default.
- SQL_INSENSITIVE = All cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor. Insensitive cursors are read-only. This corresponds to a static cursor which has a concurrency that is read-only.
- SQL_SENSITIVE = All cursors on the statement handle make visible all changes made to a result by another cursor.

SQL_ATTR_CURSOR_TYPE

A 32-bit integer value that specifies the cursor type. The supported values are:

- **SQL_CURSOR_FORWARD_ONLY** = The cursor only scrolls forward. This is the default.
- SQL_CURSOR_STATIC = The data in the result set is static.
- SQL_CURSOR_KEYSET_DRIVEN = DB2 CLI supports a pure keyset cursor. The SQL_KEYSET_SIZE statement attribute is ignored. To limit the size of the keyset the application must limit the size of the result set by setting the SQL_ATTR_MAX_ROWS attribute to a value other than 0.
- SQL_CURSOR_DYNAMIC = A dynamic scrollable cursor detects all changes (inserts, deletes and updates) to the result set, and make insertions, deletions and updates to the result set. Dynamic cursors are only supported when accessing servers which are DB2 for z/OS Version 8.1 and later.

This option cannot be specified for an open cursor.

If the specified cursor type is not supported by the data source, DB2 CLI substitutes a different cursor type and returns SQLSTATE 01S02 (Option value changed). For a mixed or dynamic cursor, DB2 CLI substitutes, in order, a keyset-driven or static cursor.

SQL_ATTR_DB2_NOBINDOUT

A Boolean attribute that specifies when and where the client performs data conversion and related tasks during a fetch operation. The default value of this attribute is 0 (false) and should only be set to 1 (true) when connected to a federated database.

SQL_ATTR_DEFERRED_PREPARE

Specifies whether the PREPARE request is deferred until the corresponding execute request is issued.

- SQL_DEFERRED_PREPARE_OFF = Disable deferred prepare. The PREPARE request will be executed the moment it is issued.
- **SQL_DEFERRED_PREPARE_ON** (default) = Enable deferred prepare. Defer the execution of the PREPARE request until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance.

  If the target DB2 database or the DDCS gateway does not support deferred prepare, the client disables deferred prepare for that connection.

Note: When deferred prepare is enabled, the row and cost estimates normally returned in the SQLERRD(3) and SQLERRD(4) of the

SQLCA of a PREPARE statement might become zeros. This might be of concern to users who want to use these values to decide whether or not to continue the SQL statement.

The default deferred prepare mode can also be set using the DEFERREDPREPARE DB2 CLI/ODBC configuration keyword.

Note: This is an IBM defined extension.

SQL_ATTR_EARLYCLOSE

Specifies whether or not the temporary cursor on the server can be automatically closed, without closing the cursor on the client, when the last record is sent to the client.

- SQL_EARLYCLOSE_OFF = Do not close the temporary cursor on the server early.
- **SQL_EARLYCLOSE_ON** = Close the temporary cursor on the server early (default).

  This saves the CLI/ODBC driver a network request by not issuing the statement to explicitly close the cursor because it knows that it has already been closed.

  Having this option on will speed up applications that make use of many small result sets.

  The EARLYCLOSE feature is not used if either:
  - The statement is disqualified for blocking.
  - The cursor type is anything other than SQL_CURSOR_FORWARD_ONLY.

Note: This is an IBM defined extension.

SQL_ATTR_ENABLE_AUTO_IPD

A 32-bit integer value that specifies whether automatic population of the IPD is performed:

- SQL_TRUE = Turns on automatic population of the IPD after a call to `SQLPrepare()`.
- SQL_FALSE = Turns off automatic population of the IPD after a call to `SQLPrepare()`.

The default value of the statement attribute SQL_ATTR_ENABLE_AUTO_IPD is equal to the value of the connection attribute SQL_ATTR_AUTO_IPD.

If the connection attribute SQL_ATTR_ AUTO_IPD is SQL_FALSE, the statement attribute SQL_ATTR_ENABLE_AUTO_IPD cannot be set to SQL_TRUE.

SQL_ATTR_FETCH_BOOKMARK_PTR

A pointer that points to a binary bookmark value. When `SQLFetchScroll()` is called with *fFetchOrientation* equal to SQL_FETCH_BOOKMARK, DB2 CLI picks up the bookmark value from this field. This field defaults to a null pointer.

SQL_ATTR_IMP_PARAM_DESC

The handle to the IPD. The value of this attribute is the descriptor allocated when the statement was initially allocated. The application cannot set this attribute.

This attribute can be retrieved by a call to `SQLGetStmtAttr()`, but not set by a call to `SQLSetStmtAttr()`.

SQL_ATTR_IMP_ROW_DESC
The handle to the IRD. The value of this attribute is the descriptor allocated when the statement was initially allocated. The application cannot set this attribute.

This attribute can be retrieved by a call to `SQLGetStmtAttr()`, but not set by a call to `SQLSetStmtAttr()`.

SQL_ATTR_INSERT_BUFFERING
This attribute enables buffering insert optimization of partitioned database environments. The possible values are:
SQL_ATTR_INSERT_BUFFERING_OFF (default),
SQL_ATTR_INSERT_BUFFERING_ON, and
SQL_ATTR_INSERT_BUFFERING_IGD (duplicates are ignored).

SQL_ATTR_KEYSET_SIZE
DB2 CLI supports a pure keyset cursor, therefore the SQL_KEYSET_SIZE statement attribute is ignored. To limit the size of the keyset the application must limit the size of the result set by setting the SQL_ATTR_MAX_ROWS attribute to a value other than 0.

SQL_ATTR_LOAD_INFO
A pointer to a structure of type db2LoadStruct. The db2LoadStruct structure is used to specify all applicable LOAD options that should be used during CLI LOAD. Note that this pointer and all of its embedded pointers should be valid during every CLI function call from the time the SQL_ATTR_USE_LOAD_API statement attribute is set to the time it is turned off. For this reason, it is recommended that this pointer and its embedded pointers point to dynamically allocated memory rather than locally declared structures.

SQL_ATTR_LOAD_ROWS_COMMITTED_PTR
A pointer to an integer that represents the total number of rows processed. This value equals the number of rows successfully loaded and committed to the database, plus the number of skipped and rejected rows. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

SQL_ATTR_LOAD_ROWS_DELETED_PTR
A pointer to an integer that represents the number of duplicate rows deleted. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

SQL_ATTR_LOAD_ROWS_LOADED_PTR
A pointer to an integer that represents the number of rows loaded into the target table. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

SQL_ATTR_LOAD_ROWS_READ_PTR
A pointer to an integer that represents the number of rows read. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

SQL_ATTR_LOAD_ROWS_REJECTED_PTR
A pointer to an integer that represents the number of rows that could not be loaded. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

SQL_ATTR_LOAD_ROWS_SKIPPED_PTR
A pointer to an integer that represents the number of rows skipped before the CLI LOAD operation began. The integer is 32-bit on 32-bit platforms and 64-bit on 64-bit platforms.

SQL_ATTR_LOB_CACHE_SIZE
A 32-bit unsigned integer that specifies maximum cache size (in bytes) for LOBs. By default, LOBs are not cached.

See the LOBCacheSize CLI/ODBC configuration keyword for further usage information.

SQL_ATTR_MAX_LENGTH
A 32-bit integer value corresponding to the maximum amount of data that can be retrieved from a single character or binary column.

Note: SQL_ATTR_MAX_LENGTH should not be used to truncate data. The *BufferLength* argument of `SQLBindCol()` or `SQLGetData()` should be used instead for truncating data.

If data is truncated because the value specified for SQL_ATTR_MAX_LENGTH is less than the amount of data available, a `SQLGetData()` call or fetch will return SQL_SUCCESS instead of returning SQL_SUCCESS_WITH_INFO and SQLSTATE **01**004 (Data Truncated). The default value for SQL_ATTR_MAX_LENGTH is **0**; 0 means that DB2 CLI will attempt to return all available data for character or binary type data.

SQL_ATTR_MAX_LOB_BLOCK_SIZE
A 32-bit unsigned integer that indicates the maximum size of LOB or XML data block. Specify a positive integer, up to 2 147 483 647. The default setting of 0 indicates that there is no limit to the data block size for LOB or XML data.

During data retrieval, the server will include all of the information for the current row in its reply to the client even if the maximum block size has been reached.

If both MaxLOBBlockSize and the db2set registry variable DB2_MAX_LOB_BLOCK_SIZE are specified, the value for MaxLOBBlockSize will be used.

Setting the MaxLOBBlockSize CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_MAX_ROWS
A 32-bit integer value corresponding to the maximum number of rows to return to the application from a query. The default value for SQL_ATTR_MAX_ROWS is **0**; 0 means all rows are returned.

SQL_ATTR_METADATA_ID
This statement attribute is defined by ODBC, but is not supported by DB2 CLI. Any attempt to set or get this attribute will result in an SQLSTATE of HYC00 (Driver not capable).

SQL_ATTR_NOSCAN
A 32-bit integer value that specifies whether DB2 CLI will scan SQL strings for escape clauses. The two permitted values are:
- **SQL_NOSCAN_OFF** - SQL strings are scanned for escape clause sequences. This is the default.
- SQL_NOSCAN_ON - SQL strings are not scanned for escape clauses. Everything is sent directly to the server for processing.

This application can choose to turn off the scanning if it never uses vendor escape sequences in the SQL strings that it sends. This will eliminate some of the overhead processing associated with scanning.

SQL_ATTR_OPTIMIZE_FOR_NROWS
A 32-bit integer value. If it is set to an integer larger than 0, ″OPTIMIZE FOR n ROWS″ clause will be appended to every select statement If set to 0 (the default) this clause will not be appended.

The default value can also be set using the OPTIMIZEFORNROWS DB2 CLI/ODBC configuration keyword.

SQL_ATTR_OPTIMIZE_SQLCOLUMNS
This attribute has been deprecated.

SQL_ATTR_PARAM_BIND_OFFSET_PTR
A 32-bit integer * value that points to an offset added to pointers to change binding of dynamic parameters. If this field is non-null, DB2 CLI dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR), and uses the resulting pointer values at execute time. It is set to null by default.

The bind offset is always added directly to the SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to a different value, the new value is added directly to the value in the descriptor field. The new offset is not added to the field value plus any earlier offsets.

Setting this statement attribute sets the SQL_DESC_BIND_OFFSET_PTR field in the APD header.

SQL_ATTR_PARAM_BIND_TYPE
A 32-bit integer value that indicates the binding orientation to be used for dynamic parameters.

This field is set to **SQL_PARAMETER_BIND_BY_COLUMN** (the default) to select column-wise binding.

To select row-wise binding, this field is set to the length of the structure or an instance of a buffer that will be bound to a set of dynamic parameters. This length must include space for all of the bound parameters and any padding of the structure or buffer to ensure that when the address of a bound parameter is incremented with the specified length, the result will point to the beginning of the same parameter in the next set of parameters. When using the sizeof operator in ANSI C, this behavior is guaranteed.

Setting this statement attribute sets the SQL_DESC_ BIND_TYPE field in the APD header.

SQL_ATTR_PARAM_OPERATION_PTR
A 16-bit unsigned integer * value that points to an array of 16-bit unsigned integer values used to specify whether or not a parameter should be ignored during execution of an SQL statement. Each value is set to either SQL_PARAM_PROCEED (for the parameter to be executed) or SQL_PARAM_IGNORE (for the parameter to be ignored).

A set of parameters can be ignored during processing by setting the status value in the array pointed to by SQL_DESC_ARRAY_STATUS_PTR in the APD to SQL_PARAM_IGNORE. A set of parameters is processed if its status value is set to SQL_PARAM_PROCEED, or if no elements in the array are set.

This statement attribute can be set to a null pointer, in which case DB2 CLI does not return parameter status values. This attribute can be set at any time, but the new value is not used until the next time `SQLExecDirect()` or `SQLExecute()` is called.

Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the APD.

SQL_ATTR_PARAM_STATUS_PTR

A 16-bit unsigned integer * value that points to an array of UWORD values containing status information for each row of parameter values after a call to `SQLExecDirect()` or `SQLExecute()`. This field is used only if SQL_ATTR_PARAMSET_SIZE is greater than 1.

The status values can contain the following values:

- SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters.
- SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters; however, warning information is available in the diagnostics data structure.
- SQL_PARAM_ERROR: There was an error in processing this set of parameters. Additional error information is available in the diagnostics data structure.
- SQL_PARAM_UNUSED: This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing.
- SQL_PARAM_DIAG_UNAVAILABLE: DB2 CLI treats arrays of parameters as a monolithic unit and so does not generate this level of error information.

This statement attribute can be set to a null pointer, in which case DB2 CLI does not return parameter status values. This attribute can be set at any time, but the new value is not used until the next time `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()` is called.

Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the IPD header.

SQL_ATTR_PARAMOPT_ATOMIC

This is a 32-bit integer value which determines, when `SQLParamOptions()` has been used to specify multiple values for parameter markers, whether the underlying processing should be done via ATOMIC or NOT-ATOMIC Compound SQL. The possible values are:

- **SQL_ATOMIC_YES** - The underlying processing makes use of ATOMIC Compound SQL. This is the default if the target database supports ATOMIC compound SQL.
- SQL_ATOMIC_NO - The underlying processing makes use of NON-ATOMIC Compound SQL.

Specifying SQL_ATOMIC_YES when connected to a server that does not support ATOMIC compound SQL results in an error (SQLSTATE is **S1**C00).

SQL_ATTR_PARAMS_PROCESSED_PTR

A 32-bit unsigned integer * record field that points to a buffer in which to return the current row number. As each row of parameters is processed, this is set to the number of that row. No row number will be returned if this is a null pointer.

Setting this statement attribute sets the SQL_DESC_ROWS_PROCESSED_PTR field in the IPD header.

If the call to `SQLExecDirect()` or `SQLExecute()` that fills in the buffer pointed to by this attribute does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.

SQL_ATTR_PARAMSET_SIZE
A 32-bit unsigned integer value that specifies the number of values for each parameter. If SQL_ATTR_PARAMSET_SIZE is greater than 1, SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR of the APD point to arrays. The cardinality of each array is equal to the value of this field.

Setting this statement attribute sets the SQL_DESC_ARRAY_SIZE field in the APD header.

SQL_ATTR_PREFETCH
This attribute has been deprecated.

SQL_ATTR_QUERY_OPTIMIZATION_LEVEL
A 32-bit integer value that sets the query optimization level to be used on the next call to `SQLPrepare()`, `SQLExtendedPrepare()`, or `SQLExecDirect()`.

Supported values to use are: -1 (default), 0, 1, 2, 3, 5, 7, and 9.

SQL_ATTR_QUERY_TIMEOUT
A 32-bit integer value that is the number of seconds to wait for an SQL statement or XQuery expression to execute before aborting the execution and returning to the application. This option can be set and used to terminate long running queries. The default value of **0** means DB2 CLI will wait indefinitely for the server to complete execution of the SQL statement. DB2 CLI supports non-zero values for all platforms that support multithreading.

When using this attribute against a server which does not have native interrupt support (such as DB2 for z/OS and OS/390, Version 7 and earlier, and DB2 UDB for iSeries), the INTERRUPT_ENABLED option must be set when cataloging the DCS database entry for the server.

When the INTERRUPT_ENABLED option is set and this attribute is set to a non-zero value, the DB2 UDB for iSeries server drops the connection and rolls back the unit of work. The application receives an SQL30081N error indicating that the connection to the server has been terminated. In order for the application to process additional database requests, the application must establish a new connection with the database server.

SQL_ATTR_REOPT
A 32-bit integer value that enables query optimization for SQL statements that contain special registers or parameter markers. Optimization occurs by using the values available at query execution time for special registers or parameter markers, instead of the default estimates that are chosen by the compiler. The valid values of the attribute are:

- 2 = SQL_REOPT_NONE. This is the default. No query optimization occurs at query execution time. The default estimates chosen by the compiler are used for the special registers or parameter markers. The default NULLID package set is used to execute dynamic SQL statements.

- 3 = SQL_REOPT_ONCE. Query optimization occurs once at query execution time, when the query is executed for the first time. The NULLIDR1 package set, which is bound with the REOPT ONCE bind option, is used.
- 4 = SQL_REOPT_ALWAYS. Query optimization or reoptimization occurs at query execution time every time the query is executed. The NULLIDRA package set, which is bound with the REOPT ALWAYS bind option, is used.

The NULLIDR1 and NULLIDRA are reserved package set names, and when used, REOPT ONCE and REOPT ALWAYS are implied respectively. These package sets have to be explicitly created with these commands:

```
db2 bind db2clipk.bnd collection NULLIDR1
db2 bind db2clipk.bnd collection NULLIDRA
```

SQL_ATTR_REOPT and SQL_ATTR_CURRENT_PACKAGE_SET are mutually exclusive, therefore, if one is set, the other is not allowed.

SQL_ATTR_RETRIEVE_DATA
A 32-bit integer value:

- **SQL_RD_ON** = `SQLFetchScroll()` and in DB2 CLI v5 and later, `SQLFetch()`, retrieve data after it positions the cursor to the specified location. This is the default.
- SQL_RD_OFF = `SQLFetchScroll()` and in DB2 CLI v5 and later, `SQLFetch()`, do not retrieve data after it positions the cursor.

By setting SQL_RETRIEVE_DATA to SQL_RD_OFF, an application can verify if a row exists or retrieve a bookmark for the row without incurring the overhead of retrieving rows.

SQL_ATTR_RETURN_USER_DEFINED_TYPES
A Boolean attribute that specifies whether user-defined type columns are reported as the user-defined type or the underlying base type when queried by functions such as `SQLDescribeCol()`. The default value is 0 (false), where columns are reported as the underlying base type.

SQL_ATTR_ROW_ARRAY_SIZE
A 32-bit integer value that specifies the number of rows in the rowset. This is the number of rows returned by each call to `SQLFetch()` or `SQLFetchScroll()`. The default value is 1.

If the specified rowset size exceeds the maximum rowset size supported by the data source, DB2 CLI substitutes that value and returns SQLSTATE 01S02 (Option value changed).

This option can be specified for an open cursor.

Setting this statement attribute sets the SQL_DESC_ARRAY_SIZE field in the ARD header.

SQL_ATTR_ROW_BIND_OFFSET_PTR
A 32-bit integer * value that points to an offset added to pointers to change binding of column data. If this field is non-null, DB2 CLI dereferences the pointer, adds the dereferenced value to each of the deferred fields in the descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR), and uses the new pointer values when binding. It is set to null by default.

Setting this statement attribute sets the SQL_DESC_BIND_OFFSET_PTR field in the ARD header.

SQL_ATTR_ROW_BIND_TYPE
A 32-bit integer value that sets the binding orientation to be used when SQLFetch() or SQLFetchScroll() is called on the associated statement. Column-wise binding is selected by supplying the defined constant SQL_BIND_BY_COLUMN in **ValuePtr*. Row-wise binding is selected by supplying a value in **ValuePtr* specifying the length of a structure or an instance of a buffer into which result columns will be bound.

The length specified in **ValuePtr* must include space for all of the bound columns and any padding of the structure or buffer to ensure that when the address of a bound column is incremented with the specified length, the result will point to the beginning of the same column in the next row. When using the **sizeof** operator with structures or unions in ANSI C, this behavior is guaranteed.

Column-wise binding is the default binding orientation for SQLFetch() and SQLFetchScroll().

Setting this statement attribute sets the SQL_DESC_BIND_TYPE field in the ARD header.

SQL_ATTR_ROW_NUMBER
A 32-bit integer value that is the number of the current row in the entire result set. If the number of the current row cannot be determined or there is no current row, DB2 CLI returns 0.

This attribute can be retrieved by a call to SQLGetStmtAttr(), but not set by a call to SQLSetStmtAttr().

SQL_ATTR_ROW_OPERATION_PTR
A 16-bit unsigned integer * value that points to an array of UDWORD values used to ignore a row during a bulk operation using SQLSetPos(). Each value is set to either SQL_ROW_PROCEED (for the row to be included in the bulk operation) or SQL_ROW_IGNORE (for the row to be excluded from the bulk operation).

This statement attribute can be set to a null pointer, in which case DB2 CLI does not return row status values. This attribute can be set at any time, but the new value is not used until the next time SQLFetch(), SQLFetchScroll(), or SQLSetPos() is called.

Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the ARD.

SQL_ATTR_ROW_STATUS_PTR
A 16-bit unsigned integer * value that points to an array of UWORD values containing row status values after a call to SQLFetch() or SQLFetchScroll(). The array has as many elements as there are rows in the rowset.

This statement attribute can be set to a null pointer, in which case DB2 CLI does not return row status values. This attribute can be set at any time, but the new value is not used until the next time SQLFetch(), SQLFetchScroll(), or SQLSetPos() is called.

Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the IRD header.

SQL_ATTR_ROWS_FETCHED_PTR
A 32-bit unsigned integer * value that points to a buffer in which to return the number of rows fetched after a call to SQLFetch() or SQLFetchScroll().

Setting this statement attribute sets the SQL_DESC_ROWS_PROCESSED_PTR field in the IRD header.

This attribute is mapped by DB2 CLI to the RowCountPtr array in a call to `SQLExtendedFetch()`.

SQL_ROWSET_SIZE
DB2 CLI applications should now use `SQLFetchScroll()` rather than `SQLExtendedFetch()`. Applications should also use the statement attribute SQL_ATTR_ROW_ARRAY_SIZE to set the number of rows in the rowset.

A 32-bit integer value that specifies the number of rows in the rowset. A rowset is the array of rows returned by each call to `SQLExtendedFetch()`. The default value is **1**, which is equivalent to making a single `SQLFetch()` call. This option can be specified even when the cursor is open and becomes effective on the next `SQLExtendedFetch()` call.

SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)
This statement attribute is not supported by DB2 CLI but is defined by ODBC.

SQL_ATTR_STMTTXN_ISOLATION
See SQL_ATTR_TXN_ISOLATION.

SQL_ATTR_STREAM_GETDATA
A 32-bit unsigned integer that indicates if the data output stream for the `SQLGetData()` function will be optimized. The values are:

- **0 (default)**: DB2 CLI buffers all the data on the client.
- 1: For applications that do not need to buffer data and are querying data on a server that supports Dynamic Data Format, specify 1 to indicate that data buffering is not required. The DB2 CLI client will optimize the data output stream.

This keyword is ignored if Dynamic Data Format is not supported by the server.

If StreamGetData is set to 1 and DB2 CLI cannot determine the number of bytes still available to return in the output buffer, `SQLGetData()` returns SQL_NO_TOTAL (-4) as the length when truncation occurs. Otherwise, `SQLGetData()` returns the number of bytes still available.

Setting the StreamGetData CLI/ODBC configuration keyword is an alternative method of specifying this behavior.

SQL_ATTR_TXN_ISOLATION
A 32-bit integer value that sets the transaction isolation level for the current *StatementHandle*.

This option cannot be set if there is an open cursor on this statement handle (SQLSTATE **24**000).

The value SQL_ATTR_STMTTXN_ISOLATION is synonymous with SQL_ATTR_TXN_ISOLATION. However, since the ODBC Driver Manager will reject the setting of SQL_ATTR_TXN_ISOLATION as a statement option, ODBC applications that need to set translation isolation level on a per statement basis must use the manifest constant SQL_ATTR_STMTTXN_ISOLATION instead on the `SQLSetStmtAttr()` call.

The default transaction isolation level can also be set using the TXNISOLATION DB2 CLI/ODBC configuration keyword.

This attribute (or corresponding keyword) is only applicable if the default isolation level is used for the statement handle. If the application has specifically set the isolation level for the statement handle, then this attribute will have no effect.

Note: It is an IBM extension to allow setting this option at the statement level.

SQL_ATTR_USE_BOOKMARKS
A 32-bit integer value that specifies whether an application will use bookmarks with a cursor:

- **SQL_UB_OFF** = Off (the default)
- SQL_UB_VARIABLE = An application will use bookmarks with a cursor, and DB2 CLI will provide variable-length bookmarks if they are supported.

To use bookmarks with a cursor, the application must specify this option with the SQL_UB_VARIABLE value before opening the cursor.

SQL_ATTR_USE_LOAD_API
A 32-bit integer that indicates if the LOAD utility will replace the regular CLI array insert for inserting data. The possible values are:

SQL_USE_LOAD_OFF
(Default) Use regular CLI array insert to insert data.

SQL_USE_LOAD_INSERT
Use the LOAD utility to append to existing data in the table.

SQL_USE_LOAD_REPLACE
Use the LOAD utility to replace existing data in the table.

SQL_USE_LOAD_RESTART
Resume a previously failed CLI LOAD operation. If the previous CLI LOAD operation failed while rows were being inserted (that is, before the SQL_ATTR_USE_LOAD_API statement attribute was set to SQL_USE_LOAD_OFF), the CLI LOAD feature will remain active, and subsequent rows will be inserted by the CLI LOAD utility. Otherwise, if the operation failed while CLI LOAD was being turned off, regular CLI array inserts will resume after the restarted load completes.

SQL_USE_LOAD_TERMINATE
Clean up and undo a previously failed CLI LOAD operation. After setting the statement attribute to this value, regular CLI array inserts will resume.

SQL_ATTR_XML_DECLARATION
A 32-bit unsigned integer that specifies which elements of an XML declaration are added to XML data when it is implicitly serialized. This attribute does not affect the result of the XMLSERIALIZE function.

This attribute can only be specified on a statement handle that has no open cursors associated with it. Attempting to update the value of this attribute while there are open cursors on the statement handle will result in a CLI0126E (SQLSTATE HY011) error, and the value remains unchanged.

Set this attribute to the sum of each component required:

0 No declarations or byte order marks (BOMs) are added to the output buffer.

1 A byte order mark (BOM) in the appropriate endianness is prepended

to the output buffer if the target encoding is UTF-16 or UTF-32. (Although a UTF-8 BOM exists, DB2 does not generate it, even if the target encoding is UTF-8.)

2 A minimal XML declaration is generated, containing only the XML version.

4 An encoding attribute that identifies the target encoding is added to any generated XML declaration. Therefore, this setting only has effect when the setting of 2 is also included when computing the value of this attribute.

Attempts to set any other value using `SQLSetStmtAttr()` or `SQLSetStmtOption()` will result in a CLI0191E (SQLSTATE HY024) error, and the value will remain unchanged.

The default setting is 7, which indicates that a BOM and an XML declaration containing the XML version and encoding attribute are generated during implicit serialization.

This attribute can also be specified on a connection handle and affects any statement handles allocated after the value is changed. Existing statement handles retain their original values.

SQL_ATTR_XQUERY_STATEMENT
A 32-bit integer value that specifies whether the statement associated with the current statement handle is an XQuery expression or an SQL statement or query. This can be used by CLI applications that do not want to prefix an XQuery expression with the "XQUERY" keyword. The supported values are:

SQL_TRUE
The next statement executed on the current statement handle is processed as an XQuery expression. If the server does not support XQuery, setting this attribute to SQL_TRUE results in a warning, CLI0005W (SQLSTATE 01S02), and the attribute's value is unchanged.

SQL_FALSE (default)
The next statement executed on the current statement handle is processed as an SQL statement.

This attribute takes effect on the next `SQLPrepare()` or `SQLExecDirect()` function call.

Related concepts:

- "Cursors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "LOB locators in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Multithreaded CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "User-defined type (UDT) usage in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "XML data handling in CLI applications - Overview" in *Call Level Interface Guide and Reference, Volume 1*
- "Buffered inserts in partitioned database environments" in *Developing SQL and External Routines*

Related tasks:

- "Specifying the rowset returned from the result set" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "appl_name - Application Name monitor element" in *System Monitor Guide and Reference*
- "SQLCA (SQL communications area)" in *SQL Reference, Volume 1*
- "ArrayInputChain CLI/ODBC configuration keyword" in *Call Level Interface Guide and Reference, Volume 1*
- "CLI/ODBC configuration keywords listing by category" in *Call Level Interface Guide and Reference, Volume 1*
- "SQLCloseCursor function (CLI) - Close cursor and discard pending results" on page 52
- "SQLDescribeCol function (CLI) - Return a set of attributes for a column" on page 82
- "SQLExecDirect function (CLI) - Execute a statement directly" on page 101
- "SQLExecute function (CLI) - Execute a statement" on page 106
- "SQLPrepare function (CLI) - Prepare a statement" on page 242
- "SQLRowCount function (CLI) - Get row count" on page 264
- "db2Load API - Load data into a table" in *Administrative API Reference*
- "SQLSetStmtAttr function (CLI) - Set options related to a statement" on page 294
- "XMLSERIALIZE scalar function" in *SQL Reference, Volume 1*
- "aslheapsz - Application support layer heap size configuration parameter" in *Performance Guide*
- "rqrioblk - Client I/O block size configuration parameter" in *Performance Guide*

Chapter 3. Descriptor FieldIdentifier and initialization values

This chapter provides a description of descriptor fields and lists the values that descriptor header and record fields are initialized to.

Descriptor FieldIdentifier argument values (CLI)

The *FieldIdentifier* argument indicates the descriptor field to be set. A descriptor contains the descriptor header, consisting of the header fields described in the next section, and zero or more descriptor records, consisting of the record fields described in the following section.

Header fields:

Each descriptor has a header consisting of the following fields.

SQL_DESC_ALLOC_TYPE [All] This read-only SQLSMALLINT header field specifies whether the descriptor was allocated automatically by DB2 CLI or explicitly by the application. The application can obtain, but not modify, this field. The field is set to SQL_DESC_ALLOC_AUTO if the descriptor was automatically allocated. It is set to SQL_DESC_ALLOC_USER if the descriptor was explicitly allocated by the application.

SQL_DESC_ARRAY_SIZE [Application descriptors] In ARDs, this SQLUINTEGER header field specifies the number of rows in the rowset. This is the number of rows to be returned by a call to `SQLFetch()`, `SQLFetchScroll()`, or `SQLSetPos()`. The default value is 1. This field can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_ROW_ARRAY_SIZE attribute.

In APDs, this SQLUINTEGER header field specifies the number of values for each parameter.

The default value of this field is 1. If SQL_DESC_ARRAY_SIZE is greater than 1, SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR of the APD or ARD point to arrays. The cardinality of each array is equal to the value of this field.

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the SQL_ROWSET_SIZE attribute. This field in the APD can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_PARAMSET_SIZE attribute.

SQL_DESC_ARRAY_STATUS_PTR [All] For each descriptor type, this SQLUSMALLINT * header field points to an array of SQLUSMALLINT values. This array is referred to as:
- row status array (IRD)
- parameter status array (IPD)
- row operation array (ARD)
- parameter operation array (APD)

In the IRD, this header field points to a row status array containing status values after a call to SQLFetch(), SQLFetchScroll(), or SQLSetPos(). The array has as many elements as there are rows in the rowset. The application must allocate an array of SQLUSMALLINTs and set this field to point to the array. The field is set to a null pointer by default. DB2 CLI will populate the array, unless the SQL_DESC_ARRAY_STATUS_PTR field is set to a null pointer, in which case no status values are generated and the array is not populated.

Note: Behavior is undefined if the application sets the elements of the row status array pointed to by the SQL_DESC_ARRAY_STATUS_PTR field of the IRD. The array is initially populated by a call to SQLFetch(), SQLFetchScroll(), or SQLSetPos(). If the call did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the array pointed to by this field are undefined.

The elements in the array can contain the following values:

- SQL_ROW_SUCCESS: The row was successfully fetched and has not changed since it was last fetched.
- SQL_ROW_SUCCESS_WITH_INFO: The row was successfully fetched and has not changed since it was last fetched. However, a warning was returned about the row.
- SQL_ROW_ERROR: An error occurred while fetching the row.
- SQL_ROW_UPDATED: The row was successfully fetched and has been updated since it was last fetched. If the row is fetched again, its status is SQL_ROW_SUCCESS.
- SQL_ROW_DELETED: The row has been deleted since it was last fetched.
- SQL_ROW_ADDED: The row was inserted by SQLSetPos(). If the row is fetched again, its status is SQL_ROW_SUCCESS.
- SQL_ROW_NOROW: The rowset overlapped the end of the result set and no row was returned that corresponded to this element of the row status array.

This field in the ARD can also be set by calling SQLSetStmtAttr() with the SQL_ATTR_ROW_STATUS_PTR attribute.

In the IPD, this header field points to a parameter status array containing status information for each set of parameter values after a call to SQLExecute() or SQLExecDirect(). If the call to SQLExecute() or SQLExecDirect() did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the array pointed to by this field are undefined. The application must allocate an array of SQLUSMALLINTs and set this field to point to the array. The driver will populate the array, unless the SQL_DESC_ARRAY_STATUS_PTR field is set to a null pointer, in which case no status values are generated and the array is not populated.

The elements in the array can contain the following values:

- SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters.
- SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters; however, warning information is available in the diagnostics data structure.
- SQL_PARAM_ERROR: An error occurred in processing this set of parameters. Additional error information is available in the diagnostics data structure.

- SQL_PARAM_UNUSED: This parameter set was unused, possibly due to the fact that some previous parameter set caused an error that aborted further processing.
- SQL_PARAM_DIAG_UNAVAILABLE: Diagnostic information is not available. An example of this is when DB2 CLI treats arrays of parameters as a monolithic unit and so does not generate this level of error information.

This field in the APD can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_PARAM_STATUS_PTR attribute.

In the ARD, this header field points to a row operation array of values that can be set by the application to indicate whether this row is to be ignored for `SQLSetPos()` operations.

The elements in the array can contain the following values:
- SQL_ROW_PROCEED: The row is included in the bulk operation using `SQLSetPos()`. (This setting does not guarantee that the operation will occur on the row. If the row has the status SQL_ROW_ERROR in the IRD row status array, DB2 CLI may not be able to perform the operation in the row.)
- SQL_ROW_IGNORE: The row is excluded from the bulk operation using `SQLSetPos()`.

If no elements of the array are set, all rows are included in the bulk operation. If the value in the SQL_DESC_ARRAY_STATUS_PTR field of the ARD is a null pointer, all rows are included in the bulk operation; the interpretation is the same as if the pointer pointed to a valid array and all elements of the array were SQL_ROW_PROCEED. If an element in the array is set to SQL_ROW_IGNORE, the value in the row status array for the ignored row is not changed.

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_ROW_OPERATION_PTR attribute.

In the APD, this header field points to a parameter operation array of values that can be set by the application to indicate whether this set of parameters is to be ignored when `SQLExecute()` or `SQLExecDirect()` is called. The elements in the array can contain the following values:
- SQL_PARAM_PROCEED: The set of parameters is included in the `SQLExecute()` or `SQLExecDirect()` call.
- SQL_PARAM_IGNORE: The set of parameters is excluded from the `SQLExecute()` or `SQLExecDirect()` call.

If no elements of the array are set, all sets of parameters in the array are used in the `SQLExecute()` or `SQLExecDirect()` calls. If the value in the SQL_DESC_ARRAY_STATUS_PTR field of the APD is a null pointer, all sets of parameters are used; the interpretation is the same as if the pointer pointed to a valid array and all elements of the array were SQL_PARAM_PROCEED.

This field in the APD can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_PARAM_OPERATION_PTR attribute.

SQL_DESC_BIND_OFFSET_PTR [Application descriptors] This SQLINTEGER * header field points to the bind offset. It is set to a null pointer by default. If this field is not a null pointer, DB2 CLI dereferences the pointer and adds the dereferenced value to each of the deferred fields that has a non-null value in the

descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR) at fetch time, and uses the new pointer values when binding.

The bind offset is always added directly to the values in the SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to a different value, the new value is still added directly to the value in each descriptor field. The new offset is not added to the field value plus any earlier offset.

This field is a *deferred field*: it is not used at the time it is set, but is used at a later time by DB2 CLI to retrieve data.

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_ROW_BIND_OFFSET_PTR attribute. This field in the ARD can also be set by calling `SQLSetStmtAttr()` with the SQL_ATTR_PARAM_BIND_OFFSET_PTR attribute.

SQL_DESC_BIND_TYPE [Application descriptors] This SQLINTEGER header field sets the binding orientation to be used for either binding columns or parameters.

In ARDs, this field specifies the binding orientation when `SQLFetchScroll()` is called on the associated statement handle.

To select column-wise binding for columns, this field is set to SQL_BIND_BY_COLUMN (the default).

This field in the ARD can also be set by calling `SQLSetStmtAttr()` with SQL_ATTR_ROW_BIND_TYPE Attribute.

In APDs, this field specifies the binding orientation to be used for dynamic parameters.

To select column-wise binding for parameters, this field is set to SQL_BIND_BY_COLUMN (the default).

This field in the APD can also be set by calling `SQLSetStmtAttr()` with SQL_ATTR_PARAM_BIND_TYPE Attribute.

SQL_DESC_COUNT [All] This SQLSMALLINT header field specifies the one-based index of the highest-numbered record that contains data. When DB2 CLI sets the data structure for the descriptor, it must also set the COUNT field to show how many records are significant. When an application allocates an instance of this data structure, it does not have to specify how many records to reserve room for. As the application specifies the contents of the records, DB2 CLI takes any required action to ensure that the descriptor handle refers to a data structure of adequate size.

SQL_DESC_COUNT is not a count of all data columns that are bound (if the field is in an ARD), or all parameters that are bound (in an APD), but the number of the highest-numbered record. If a column or a parameter with a number that is less than the number of the highest-numbered column is unbound (by calling `SQLBindCol()` with the *Target ValuePtr* argument set to a null pointer, or `SQLBindParameter()` with the *Parameter ValuePtr* argument set to a null pointer), SQL_DESC_COUNT is not changed. If additional columns or parameters are

bound with numbers greater than the highest-numbered record that contains data, DB2 CLI automatically increases the value in the SQL_DESC_COUNT field. If all columns or parameters are unbound by calling SQLFreeStmt() with the SQL_UNBIND option, SQL_DESC_COUNT is set to 0.

The value in SQL_DESC_COUNT can be set explicitly by an application by calling SQLSetDescField(). If the value in SQL_DESC_COUNT is explicitly decreased, all records with numbers greater than the new value in SQL_DESC_COUNT are removed, unbinding the columns. If the value in SQL_DESC_COUNT is explicitly set to 0, and the field is in an APD, all parameters are unbound. If the value in SQL_DESC_COUNT is explicitly set to 0, and the field is in an ARD, all data buffers except a bound bookmark column are released.

The record count in this field of an ARD does not include a bound bookmark column.

SQL_DESC_ROWS_PROCESSED_PTR [Implementation descriptors] In an IRD, this SQLUINTEGER * header field points to a buffer containing the number of rows fetched after a call to SQLFetch() or SQLFetchScroll(), or the number of rows affected in a bulk operation performed by a call to SQLSetPos().

In an IPD, this SQLUINTEGER * header field points to a buffer containing the number of the row as each row of parameters is processed. No row number will be returned if this is a null pointer.

SQL_DESC_ROWS_PROCESSED_PTR is valid only after SQL_SUCCESS or SQL_SUCCESS_WITH_INFO has been returned after a call to SQLFetch() or SQLFetchScroll() (for an IRD field) or SQLExecute() or SQLExecDirect() (for an IPD field). If the return code is not one of the above, the location pointed to by SQL_DESC_ROWS_PROCESSED_PTR is undefined. If the call that fills in the buffer pointed to by this field did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined, unless it returns SQL_NO_DATA, in which case the value in the buffer is set to 0.

This field in the ARD can also be set by calling SQLSetStmtAttr() with the SQL_ATTR_ROWS_FETCHED_PTR attribute. This field in the ARD can also be set by calling SQLSetStmtAttr() with the SQL_ATTR_PARAMS_PROCESSED_PTR attribute.

The buffer pointed to by this field is allocated by the application. It is a deferred output buffer that is set by DB2 CLI. It is set to a null pointer by default.

Record fields:

Each descriptor contains one or more records consisting of fields that define either column data or dynamic parameters, depending on the type of descriptor. Each record is a complete definition of a single column or parameter.

SQL_DESC_AUTO_UNIQUE_VALUE [IRDs] This read-only SQLINTEGER record field contains SQL_TRUE if the column is an auto-incrementing column, or SQL_FALSE if the column is not an auto-incrementing column. This field is read-only, but the underlying auto-incrementing column is not necessarily read-only.

SQL_DESC_BASE_COLUMN_NAME [IRDs] This read-only SQLCHAR record field contains the base column name for the result set column. If a base column name does not exist (as in the case of columns that are expressions), then this variable contains an empty string.

SQL_DESC_BASE_TABLE_NAME [IRDs] This read-only SQLCHAR record field contains the base table name for the result set column. If a base table name cannot be defined or is not applicable, then this variable contains an empty string.

SQL_DESC_CASE_SENSITIVE [Implementation descriptors] This read-only SQLINTEGER record field contains SQL_TRUE if the column or parameter is treated as case-sensitive for collations and comparisons, or SQL_FALSE if the column is not treated as case-sensitive for collations and comparisons, or if it is a non-character column.

SQL_DESC_CATALOG_NAME [IRDs] This read-only SQLCHAR record field contains the catalog or qualifier name for the base table that contains the column. The return value is driver-dependent if the column is an expression or if the column is part of a view. If the data source does not support catalogs (or qualifiers) or the catalog or qualifier name cannot be determined, this variable contains an empty string.

SQL_DESC_CONCISE_TYPE [All] This SQLSMALLINT header field specifies the concise data type for all data types.

The values in the SQL_DESC_CONCISE_TYPE and SQL_DESC_TYPE fields are interdependent. Each time one of the fields is set, the other must also be set. SQL_DESC_CONCISE_TYPE can be set by a call to `SQLBindCol()` or `SQLBindParameter()`, or `SQLSetDescField()`. SQL_DESC_TYPE can be set by a call to `SQLSetDescField()` or `SQLSetDescRec()`.

If SQL_DESC_CONCISE_TYPE is set to a concise data type, SQL_DESC_TYPE field is set to the same value, and the SQL_DESC_DATETIME_INTERVAL_CODE field is set to 0.

SQL_DESC_DATA_PTR [Application descriptors and IPDs] This SQLPOINTER record field points to a variable that will contain the parameter value (for APDs) or the column value (for ARDs). The descriptor record (and either the column or parameter that it represents) is unbound if Target*ValuePtr* in a call to either `SQLBindCol()` or `SQLBindParameter()` is a null pointer, or the SQL_DESC_DATA_PTR field in a call to `SQLSetDescField()` or `SQLSetDescRec()` is set to a null pointer. Other fields are not affected if the SQL_DESC_DATA_PTR field is set to a null pointer. If the call to `SQLFetch()` or `SQLFetchScroll()` that fills in the buffer pointed to by this field did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.

This field is a deferred field: it is not used at the time it is set, but is used at a later time by DB2 CLI to retrieve data.

Whenever the SQL_DESC_DATA_PTR field is set, DB2 CLI checks that the value in the SQL_DESC_TYPE field contains valid DB2 CLI or ODBC data types, and that all other fields affecting the data types are consistent. Refer to the consistency checks information for more detail.

SQL_DESC_DATETIME_INTERVAL_CODE [All] This SQLSMALLINT record field contains the subcode for the specific datetime data type when the SQL_DESC_TYPE field is SQL_DATETIME. This is true for both SQL and C data types.

This field can be set to the following for datetime data types:

Table 153. Datetime subcodes

Datetime types	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME/SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIMESTAMP/ SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

ODBC 3.0 defines other values (not listed here) for interval data types, which DB2 CLI does not support. If any other value is specified in a `SQLSetDescRec()` or `SQLSetDescField()` call, an error will be generated indicating HY092 (Option type out of range).

SQL_DESC_DATETIME_INTERVAL_PRECISION [All] ODBC 3.0 defines this SQLINTEGER record field, however, DB2 CLI does not support interval data types. The fixed value returned is 0. Any attempt to set this field will result in 01S02 (Option value changed).

SQL_DESC_DISPLAY_SIZE [IRDs] This read-only SQLINTEGER record field contains the maximum number of characters required to display the data from the column. The value in this field is not the same as the descriptor field SQL_DESC_LENGTH because the LENGTH field is undefined for all numeric types.

SQL_DESC_FIXED_PREC_SCALE [Implementation descriptors] This read-only SQLSMALLINT record field is set to SQL_TRUE if the column is an exact numeric column and has a fixed precision and non-zero scale, or SQL_FALSE if the column is not an exact numeric column with a fixed precision and scale.

SQL_DESC_INDICATOR_PTR [Application descriptors] In ARDs, this SQLINTEGER * record field points to the indicator variable. This variable contains SQL_NULL_DATA if the column value is NULL. For APDs, the indicator variable is set to SQL_NULL_DATA to specify NULL dynamic arguments. Otherwise, the variable is zero (unless the values in SQL_DESC_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PTR are the same pointer).

If the SQL_DESC_INDICATOR_PTR field in an ARD is a null pointer, DB2 CLI is prevented from returning information about whether the column is NULL or not. If the column is NULL and INDICATOR_PTR is a null pointer, SQLSTATE 22002, "Indicator variable required but not supplied," is returned when DB2 CLI attempts to populate the buffer after a call to `SQLFetch()` or `SQLFetchScroll()`. If the call to `SQLFetch()` or `SQLFetchScroll()` did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.

The SQL_DESC_INDICATOR_PTR field determines whether the field pointed to by SQL_DESC_OCTET_LENGTH_PTR is set. If the data value for a column is NULL, DB2 CLI sets the indicator variable to SQL_NULL_DATA. The field pointed to by SQL_DESC_OCTET_LENGTH_PTR is then not set. If a NULL value is not encountered during the fetch, the buffer pointed to by

SQL_DESC_INDICATOR_PTR is set to zero, and the buffer pointed to by SQL_DESC_OCTET_LENGTH_PTR is set to the length of the data.

If the INDICATOR_PTR field in an APD is a null pointer, the application cannot use this descriptor record to specify NULL arguments.

This field is a deferred field: it is not used at the time it is set, but is used at a later time by DB2 CLI to store data.

SQL_DESC_LABEL [IRDs] This read-only SQLCHAR record field contains the column label or title. If the column does not have a label, this variable contains the column name. If the column is unnamed and unlabeled, this variable contains an empty string.

SQL_DESC_LENGTH [All] This SQLUINTEGER record field is either the maximum or actual character length of a character string or a binary data type. It is the maximum character length for a fixed-length data type, or the actual character length for a variable-length data type. Its value always excludes the null termination character that ends the character string. Note that this field is a count of characters, not a count of bytes.

SQL_DESC_LITERAL_PREFIX [IRDs] This read-only SQLCHAR record field contains the character or characters that DB2 CLI recognizes as a prefix for a literal of this data type. This variable contains an empty string for a data type for which a literal prefix is not applicable.

SQL_DESC_LITERAL_SUFFIX [IRDs] This read-only SQLCHAR record field contains the character or characters that DB2 CLI recognizes as a suffix for a literal of this data type. This variable contains an empty string for a data type for which a literal suffix is not applicable.

SQL_DESC_LOCAL_TYPE_NAME [Implementation descriptors] This read-only SQLCHAR record field contains any localized (native language) name for the data type that may be different from the regular name of the data type. If there is no localized name, then an empty string is returned. This field is for display purposes only.

SQL_DESC_NAME [Implementation descriptors] This SQLCHAR record field in a row descriptor contains the column alias, if it applies. If the column alias does not apply, the column name is returned. In either case, the UNNAMED field is set to SQL_NAMED. If there is no column name or a column alias, an empty string is returned in the NAME field and the UNNAMED field is set to SQL_UNNAMED.

An application can set the SQL_DESC_NAME field of an IPD to a parameter name or alias to specify stored procedure parameters by name. The SQL_DESC_NAME field of an IRD is a read-only field; SQLSTATE HY091 (Invalid descriptor field identifier) will be returned if an application attempts to set it.

In IPDs, this field is undefined if dynamic parameters are not supported. If named parameters are supported and the version of DB2 CLI is capable of describing parameters, then the parameter name is returned in this field.

The column name value can be affected by the environment attribute SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA set by `SQLSetEnvAttr()`.

SQL_DESC_NULLABLE [Implementation descriptors] In IRDs, this read-only SQLSMALLINT record field is SQL_NULLABLE if the column can have NULL values; SQL_NO_NULLS if the column cannot have NULL values; or SQL_NULLABLE_UNKNOWN if it is not known whether the column accepts NULL values. This field pertains to the result set column, not the base column.

In IPDs, this field is always set to SQL_NULLABLE, since dynamic parameters are always nullable, and cannot be set by an application.

SQL_DESC_NUM_PREC_RADIX [All] This SQLINTEGER field contains a value of 2 if the data type in the SQL_DESC_TYPE field is an approximate numeric data type, because the SQL_DESC_PRECISION field contains the number of bits. This field contains a value of 10 if the data type in the SQL_DESC_TYPE field is an exact numeric data type, because the SQL_DESC_PRECISION field contains the number of decimal digits. This field is set to 0 for all non-numeric data types.

SQL_DESC_OCTET_LENGTH [All] This SQLINTEGER record field contains the length, in bytes, of a character string or binary data type. For fixed-length character types, this is the actual length in bytes. For variable-length character or binary types, this is the maximum length in bytes. This value always excludes space for the null termination character for implementation descriptors and always includes space for the null termination character for application descriptors. For application data, this field contains the size of the buffer. For APDs, this field is defined only for output or input/output parameters.

SQL_DESC_OCTET_LENGTH_PTR [Application descriptors] This SQLINTEGER * record field points to a variable that will contain the total length in bytes of a dynamic argument (for parameter descriptors) or of a bound column value (for row descriptors).

For an APD, this value is ignored for all arguments except character string and binary; if this field points to SQL_NTS, the dynamic argument must be null-terminated. To indicate that a bound parameter will be a data-at-execute parameter, an application sets this field in the appropriate record of the APD to a variable that, at execute time, will contain the value SQL_DATA_AT_EXEC. If there is more than one such field, SQL_DESC_DATA_PTR can be set to a value uniquely identifying the parameter to help the application determine which parameter is being requested.

If the OCTET_LENGTH_PTR field of an ARD is a null pointer, DB2 CLI does not return length information for the column. If the SQL_DESC_OCTET_LENGTH_PTR field of an APD is a null pointer, DB2 CLI assumes that character strings and binary values are null terminated. (Binary values should not be null terminated, but should be given a length, in order to avoid truncation.)

If the call to `SQLFetch()` or `SQLFetchScroll()` that fills in the buffer pointed to by this field did not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the contents of the buffer are undefined.

This field is a deferred field: it is not used at the time it is set, but is used at a later time by DB2 CLI to buffer data.

By default this is a pointer to a 4-byte value.

SQL_DESC_PARAMETER_TYPE [IPDs] This SQLSMALLINT record field is set to SQL_PARAM_INPUT for an input parameter, SQL_PARAM_INPUT_OUTPUT for an input/output parameter, or SQL_PARAM_OUTPUT for an output parameter. Set to SQL_PARAM_INPUT by default.

For an IPD, the field is set to SQL_PARAM_INPUT by default if the IPD is not automatically populated by DB2 CLI (the SQL_ATTR_ENABLE_AUTO_IPD statement attribute is SQL_FALSE). An application should set this field in the IPD for parameters that are not input parameters.

SQL_DESC_PRECISION [All] This SQLSMALLINT record field contains the number of digits for an exact numeric type, the number of bits in the mantissa (binary precision) for an approximate numeric type, or the numbers of digits in the fractional seconds component for the SQL_TYPE_TIME or SQL_TYPE_TIMESTAMP data types. This field is undefined for all other data types.

SQL_DESC_SCALE [All] This SQLSMALLINT record field contains the defined scale for DECIMAL and NUMERIC data types. The field is undefined for all other data types.

SQL_DESC_SCHEMA_NAME [IRDs] This read-only SQLCHAR record field contains the schema name of the base table that contains the column. For many DBMS's, this is the owner name. If the data source does not support schemas (or owners) or the schema name cannot be determined, this variable contains an empty string.

SQL_DESC_SEARCHABLE [IRDs] This read-only SQLSMALLINT record field is set to one of the following values:

- SQL_PRED_NONE if the column cannot be used in a WHERE clause. (This is the same as the SQL_UNSEARCHABLE value defined in ODBC 2.0.)
- SQL_PRED_CHAR if the column can be used in a WHERE clause, but only with the LIKE predicate. (This is the same as the SQL_LIKE_ONLY value defined in ODBC 2.0.)
- SQL_PRED_BASIC if the column can be used in a WHERE clause with all the comparison operators except LIKE. (This is the same as the SQL_EXCEPT_LIKE value defined in ODBC 2.0.)
- SQL_PRED_SEARCHABLE if the column can be used in a WHERE clause with any comparison operator.

SQL_DESC_TABLE_NAME [IRDs] This read-only SQLCHAR record field contains the name of the base table that contains this column.

SQL_DESC_TYPE [All] This SQLSMALLINT record field specifies the concise SQL or C data type for all data types.

Note: ODBC 3.0 defines the SQL_INTERVAL data type which is not supported by DB2 CLI. Any behavior associated with this data type is not present in DB2 CLI.

The values in the SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE fields are interdependent. Each time one of the fields is set, the other must also be set. SQL_DESC_TYPE can be set by a call to `SQLSetDescField()` or `SQLSetDescRec()`. SQL_DESC_CONCISE_TYPE can be set by a call to `SQLBindCol()` or `SQLBindParameter()`, or `SQLSetDescField()`.

If SQL_DESC_TYPE is set to a concise data type, the SQL_DESC_CONCISE_TYPE field is set to the same value, and the SQL_DESC_DATETIME_INTERVAL_CODE field is set to 0.

When the SQL_DESC_TYPE field is set by a call to `SQLSetDescField()`, the following fields are set to the following default values. The values of the remaining fields of the same record are undefined:

Table 154. Default values

SQL_DESC_TYPE	Other fields Implicitly Set
SQL_CHAR, SQL_VARCHAR	SQL_DESC_LENGTH is set to 1. SQL_DESC_PRECISION is set to 0.
SQL_DECIMAL, SQL_NUMERIC	SQL_DESC_SCALE is set to 0. SQL_DESC_PRECISION is set to the precision for the respective data type.
SQL_FLOAT	SQL_DESC_PRECISION is set to the default precision for SQL_FLOAT.
SQL_DATETIME	SQL_DESC_CONCISE_TYPE and/or SQL_DESC_DATETIME_INTERVAL_CODE may be set implicitly to indicate a DATE SQL or C type.
SQL_INTERVAL	This data type is not supported by DB2 CLI.

When an application calls `SQLSetDescField()` to set fields of a descriptor, rather than calling `SQLSetDescRec()`, the application must first declare the data type. If the values implicitly set are unacceptable, the application can then call `SQLSetDescField()` to set the unacceptable value explicitly.

SQL_DESC_TYPE_NAME [Implementation descriptors] This read-only SQLCHAR record field contains the data-source-dependent type name (for example, “CHAR”, “VARCHAR”, and so on). If the data type name is unknown, this variable contains an empty string.

SQL_DESC_UNNAMED [Implementation descriptors] This SQLSMALLINT record field in a row descriptor is set to either SQL_NAMED or SQL_UNNAMED. If the NAME field contains a column alias, or if the column alias does not apply, the UNNAMED field is set to SQL_NAMED. If there is no column name or a column alias, the UNNAMED field is set to SQL_UNNAMED.

An application can set the SQL_DESC_UNNAMED field of an IPD to SQL_UNNAMED. SQLSTATE HY091 (Invalid descriptor field identifier) is returned if an application attempts to set the SQL_DESC_UNNAMED field of an IPD to SQL_NAMED. The SQL_DESC_UNNAMED field of an IRD is read-only; SQLSTATE HY091 (Invalid descriptor field identifier) will be returned if an application attempts to set it.

SQL_DESC_UNSIGNED [Implementation descriptors] This read-only SQLSMALLINT record field is set to SQL_TRUE if the column type is unsigned or non-numeric, or SQL_FALSE if the column type is signed.

SQL_DESC_UPDATABLE [IRDs] This read-only SQLSMALLINT record field is set to one of the following values:
- SQL_ATTR_READ_ONLY if the result set column is read-only.
- SQL_ATTR_WRITE if the result set column is read-write.
- SQL_ATTR_READWRITE_UNKNOWN if it is not known whether the result set column is updatable or not.

SQL_DESC_UPDATABLE describes the updatability of the column in the result set, not the column in the base table. The updatability of the column in the base table on which this result set column is based may be different than the value in this field. Whether a column is updatable can be based on the data type, user privileges, and the definition of the result set itself. If it is unclear whether a column is updatable, SQL_UPDT_READWRITE_UNKNOWN should be returned.

SQL_DESC_USER_DEFINED_TYPE_CODE [IRDs] This read-only SQLINTEGER returns information that describes the nature of a column's data type. Four values may be returned:

- SQL_TYPE_BASE: the column data type is a base data type, such as CHAR, DATE, or DOUBLE).
- SQL_TYPE_DISTINCT: the column data type is a distinct user-defined type.
- SQL_TYPE_REFERENCE: the column data type is a reference user-defined type.
- SQL_TYPE_STRUCTURED: the column data type is a structured user-defined type.

Related concepts:

- "Consistency checks for descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "Descriptor header and record field initialization values (CLI)" on page 378
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" on page 160
- "SQLGetDescRec function (CLI) - Get multiple field settings of descriptor record" on page 164
- "SQLSetDescField function (CLI) - Set a single field of a descriptor record" on page 276
- "SQLSetEnvAttr function (CLI) - Set environment attribute" on page 284
- "User-defined types" in *SQL Reference, Volume 1*

Descriptor header and record field initialization values (CLI)

The following tables list the initialization of each field for each type of descriptor, with "D" indicating that the field is initialized with a default, and "ND" indicating that the field is initialized without a default. If a number is shown, the default value of the field is that number. The tables also indicate whether a field is read/write (R/W) or read-only (R).

The initialization of header fields is as follows:

Table 155. Initialization of header fields

Descriptor header field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_ALLOC_TYPE	SQLSMALLINT	• ARD: R • APD: R • IRD: R • IPD: R	• ARD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit • APD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit • IRD: SQL_DESC_ALLOC_AUTO • IPD: SQL_DESC_ALLOC_AUTO
SQL_DESC_ARRAY_SIZE	SQLUINTEGER	• ARD: R/W • APD: R/W • IRD: Unused • IPD: Unused	• ARD: [a] • APD: [a] • IRD: Unused • IPD: Unused
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT *	• ARD: R/W • APD: R/W • IRD: R/W • IPD: R/W	• ARD: Null ptr • APD: Null ptr • IRD: Null ptr • IPD: Null ptr
SQL_DESC_BIND_OFFSET_PTR	SQLINTEGER *	• ARD: R/W • APD: R/W • IRD: Unused • IPD: Unused	• ARD: Null ptr • APD: Null ptr • IRD: Unused • IPD: Unused
SQL_DESC_BIND_TYPE	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: Unused • IPD: Unused	• ARD: SQL_BIND_BY_COLUMN • APD: SQL_BIND_BY_COLUMN • IRD: Unused • IPD: Unused
SQL_DESC_COUNT	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: 0 • APD: 0 • IRD: D • IPD: 0
SQL_DESC_ROWS_ PROCESSED_PTR	SQLUINTEGER *	• ARD: Unused • APD: Unused • IRD: R/W • IPD: R/W	• ARD: Unused • APD: Unused • IRD: Null Ptr • IPD: Null Ptr

a These fields are defined only when the IPD is automatically populated by DB2 CLI. If the fields are not automatically populated then they are undefined. If an application attempts to set these fields, SQLSTATE HY091 (Invalid descriptor field identifier.) will be returned.

The initialization of record fields is as follows:

Table 156. Initialization of record fields

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused

Table 156. Initialization of record fields (continued)

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	• ARD: Unused • APD: Unused • IRD: R • IPD: R	• ARD: Unused • APD: Unused • IRD: D • IPD: D [a]
SQL_DESC_CATALOG_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: SQL_C_DEFAULT • APD: SQL_C_DEFAULT • IRD: D • IPD: ND
SQL_DESC_DATA_PTR	SQLPOINTER	• ARD: R/W • APD: R/W • IRD: Unused • IPD: Unused	• ARD: Null ptr • APD: Null ptr • IRD: Unused • IPD: Unused [b]
SQL_DESC_DATETIME_ INTERVAL_CODE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_DATETIME_ INTERVAL_PRECISION	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_DISPLAY_SIZE	SQLINTEGER	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	• ARD: Unused • APD: Unused • IRD: R • IPD: R	• ARD: Unused • APD: Unused • IRD: D • IPD: D [a]
SQL_DESC_INDICATOR_PTR	SQLINTEGER *	• ARD: R/W • APD: R/W • IRD: Unused • IPD: Unused	• ARD: Null ptr • APD: Null ptr • IRD: Unused • IPD: Unused
SQL_DESC_LABEL	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused

Table 156. Initialization of record fields (continued)

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_LENGTH	SQLUINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: R	• ARD: Unused • APD: Unused • IRD: D • IPD: D [a]
SQL_DESC_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_NULLABLE	SQLSMALLINT	• ARD: Unused • APD: Unused • IRD: R • IPD: R	• ARD: ND • APD: ND • IRD: N • IPD: ND
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_OCTET_LENGTH	SQLINTEGER	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_OCTET_LENGTH_PTR	SQLINTEGER *	• ARD: R/W • APD: R/W • IRD: Unused • IPD: Unused	• ARD: Null ptr • APD: Null ptr • IRD: Unused • IPD: Unused
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	• ARD: Unused • APD: Unused • IPD: Unused • IRD: R/W	• ARD: Unused • APD: Unused • IPD: Unused • IRD: D=SQL_PARAM_INPUT
SQL_DESC_PRECISION	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_SCALE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND

Table 156. Initialization of record fields (continued)

Descriptor record field	Type	Readable and writable (R/W) or read-only (R)	Initialization value
SQL_DESC_SCHEMA_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_SEARCHABLE	SQLSMALLINT	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_TABLE_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused
SQL_DESC_TYPE	SQLSMALLINT	• ARD: R/W • APD: R/W • IRD: R • IPD: R/W	• ARD: SQL_C_DEFAULT • APD: SQL_C_DEFAULT • IRD: D • IPD: ND
SQL_DESC_TYPE_NAME	SQLCHAR *	• ARD: Unused • APD: Unused • IRD: R • IPD: R	• ARD: Unused • APD: Unused • IRD: D • IPD: D [a]
SQL_DESC_UNNAMED	SQLSMALLINT	• ARD: Unused • APD: Unused • IRD: R • IPD: R/W	• ARD: ND • APD: ND • IRD: D • IPD: ND
SQL_DESC_UNSIGNED	SQLSMALLINT	• ARD: Unused • APD: Unused • IRD: R • IPD: R	• ARD: Unused • APD: Unused • IRD: D • IPD: D [a]
SQL_DESC_UPDATABLE	SQLSMALLINT	• ARD: Unused • APD: Unused • IRD: R • IPD: Unused	• ARD: Unused • APD: Unused • IRD: D • IPD: Unused

a These fields are defined only when the IPD is automatically populated by DB2 CLI. If the fields are not automatically populated then they are undefined. If an application attempts to set these fields, SQLSTATE HY091 (Invalid descriptor field identifier.) will be returned.

b The SQL_DESC_DATA_PTR field in the IPD can be set to force a consistency check. In a subsequent call to `SQLGetDescField()` or `SQLGetDescRec()`, DB2 CLI is not required to return the value that SQL_DESC_DATA_PTR was set to.

Related concepts:

- "Consistency checks for descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptors in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "C data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*
- "Descriptor FieldIdentifier argument values (CLI)" on page 367
- "SQLGetDescField function (CLI) - Get single field settings of descriptor record" on page 160
- "SQLSetDescField function (CLI) - Set a single field of a descriptor record" on page 276

Chapter 4. DiagIdentifier argument values

The `DiagIdentifier` argument indicates the field of the diagnostic data structure to be retrieved. This chapter describes the possible header and record fields.

Header and record fields for the DiagIdentifier argument (CLI)

Header fields

The following header fields can be included in the *DiagIdentifier* argument. The only diagnostic header fields that are defined for a descriptor field are SQL_DIAG_NUMBER and SQL_DIAG_RETURNCODE.

Table 157. Header fields for DiagIdentifier arguments

Header fields	Return type	Description
SQL_DIAG_CURSOR_ROW_ COUNT	SQLINTEGER	This field contains the count of rows in the cursor. Its semantics depend upon the `SQLGetInfo()` information types: • SQL_DYNAMIC_CURSOR_ ATTRIBUTES2 • SQL_FORWARD_ONLY_ CURSOR_ATTRIBUTES2 • SQL_KEYSET_CURSOR_ ATTRIBUTES2 • SQL_STATIC_CURSOR_ ATTRIBUTES2 which indicate which row counts are available for each cursor type (in the SQL_CA2_CRC_EXACT and SQL_CA2_CRC_APPROXIMATE bits). The contents of this field are defined only for statement handles and only after `SQLExecute()`, `SQLExecDirect()`, or `SQLMoreResults()` has been called. Calling `SQLGetDiagField()` with a *DiagIdentifier* of SQL_DIAG_CURSOR_ROW_COUNT on a handle other than a statement handle will return SQL_ERROR.
SQL_DIAG_DYNAMIC_ FUNCTION	CHAR *	This is a string that describes the SQL statement that the underlying function executed (see “Dynamic function fields” on page 388 for the values that DB2 CLI supports). The contents of this field are defined only for statement handles, and only after a call to `SQLExecute()` or `SQLExecDirect()`. The value of this field is undefined before a call to `SQLExecute()` or `SQLExecDirect()`.

Table 157. Header fields for DiagIdentifier arguments (continued)

Header fields	Return type	Description
SQL_DIAG_DYNAMIC_ FUNCTION_CODE	SQLINTEGER	This is a numeric code that describes the SQL statement that was executed by the underlying function (see "Dynamic function fields" on page 388 for the values that DB2 CLI supports). The contents of this field are defined only for statement handles, and only after a call to `SQLExecute()` or `SQLExecDirect()`. The value of this field is undefined before a call to `SQLExecute()`, `SQLExecDirect()`, or `SQLMoreResults()`. Calling `SQLGetDiagField()` with a *DiagIdentifier* of SQL_DIAG_DYNAMIC_FUNCTION_CODE on a handle other than a statement handle will return SQL_ERROR. The value of this field is undefined before a call to `SQLExecute()` or `SQLExecDirect()`.
SQL_DIAG_NUMBER	SQLINTEGER	The number of status records that are available for the specified handle.
SQL_DIAG_RELATIVE_ COST_ESTIMATE	SQLINTEGER	If `SQLPrepare()` is invoked and successful, contains a relative cost estimate of the resources required to process the statement. If deferred prepare is enabled, this field will have the value of 0 until the statement is executed.
SQL_DIAG_RETURNCODE	RETCODE	Return code returned by the last executed function associated with the specified handle. If no function has yet been called on the *Handle*, SQL_SUCCESS will be returned for SQL_DIAG_RETURNCODE.
SQL_DIAG_ROW_COUNT	SQLINTEGER	The number of rows affected by an insert, delete, or update performed by `SQLExecute()`, `SQLExecDirect()`, or `SQLSetPos()`. It is defined after a cursor specification has been executed. The contents of this field are defined only for statement handles. The data in this field is returned in the *RowCountPtr* argument of `SQLRowCount()`. The data in this field is reset after every function call, whereas the row count returned by `SQLRowCount()` remains the same until the statement is set back to the prepared or allocated state.

Record fields

The following record fields can be included in the *DiagIdentifier* argument:

Table 158. Record fields for DiagIdentifier arguments

Record fields	Return type	Descritpion
SQL_DIAG_CLASS_ORIGIN	CHAR *	A string that indicates the document that defines the class and subclass portion of the SQLSTATE value in this record. DB2 CLI always returns an empty string for SQL_DIAG_CLASS_ORIGIN.
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	If the SQL_DIAG_ROW_NUMBER field is a valid row number in a rowset or set of parameters, then this field contains the value that represents the column number in the result set. Result set column numbers always start at 1; if this status record pertains to a bookmark column, then the field can be zero. It has the value SQL_NO_COLUMN_NUMBER if the status record is not associated with a column number. If DB2 CLI cannot determine the column number that this record is associated with, this field has the value SQL_COLUMN_NUMBER_UNKNOWN. The contents of this field are defined only for statement handles.
SQL_DIAG_CONNECTION_NAME	CHAR *	A string that indicates the name of the connection that the diagnostic record relates to. DB2 CLI always returns an empty string for SQL_DIAG_CONNECTION_NAME
SQL_DIAG_MESSAGE_TEXT	CHAR *	An informational message on the error or warning.
SQL_DIAG_NATIVE	SQLINTEGER	A driver/data-source-specific native error code. If there is no native error code, the driver returns 0.
SQL_DIAG_ROW_NUMBER	SQLINTEGER	This field contains the row number in the rowset, or the parameter number in the set of parameters, with which the status record is associated. This field has the value SQL_NO_ROW_NUMBER if this status record is not associated with a row number. If DB2 CLI cannot determine the row number that this record is associated with, this field has the value SQL_ROW_NUMBER_UNKNOWN. The contents of this field are defined only for statement handles.
SQL_DIAG_SERVER_NAME	CHAR *	A string that indicates the server name that the diagnostic record relates to. It is the same as the value returned for a call to `SQLGetInfo()` with the SQL_DATA_SOURCE_NAME *InfoType*. For diagnostic data structures associated with the environment handle and for diagnostics that do not relate to any server, this field is a zero-length string.
SQL_DIAG_SQLSTATE	CHAR *	A five-character SQLSTATE diagnostic code.

Table 158. Record fields for DiagIdentifier arguments (continued)

Record fields	Return type	Descritpion
SQL_DIAG_SUBCLASS_ORIGIN	CHAR *	A string with the same format and valid values as SQL_DIAG_CLASS_ORIGIN, that identifies the defining portion of the subclass portion of the SQLSTATE code.

Values of the dynamic function fields

The table below describes the values of SQL_DIAG_DYNAMIC_FUNCTION and SQL_DIAG_DYNAMIC_FUNCTION_CODE that apply to each type of SQL statement executed by a call to `SQLExecute()` or `SQLExecDirect()`. This is the list that DB2 CLI uses. ODBC also specifies other values.

Table 159. Values of dynamic function fields

SQL statement executed	Value of SQL_DIAG_ DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_ FUNCTION_CODE
alter-table-statement	"ALTER TABLE"	SQL_DIAG_ALTER_TABLE
create-index-statement	"CREATE INDEX"	SQL_DIAG_CREATE_INDEX
create-table-statement	"CREATE TABLE"	SQL_DIAG_CREATE_TABLE
create-view-statement	"CREATE VIEW"	SQL_DIAG_CREATE_VIEW
cursor-specification	"SELECT CURSOR"	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	"DYNAMIC DELETE CURSOR"	SQL_DIAG_DYNAMIC_DELETE_ CURSOR
delete-statement-searched	"DELETE WHERE"	SQL_DIAG_DELETE_WHERE
drop-index-statement	"DROP INDEX"	SQL_DIAG_DROP_INDEX
drop-table-statement	"DROP TABLE"	SQL_DIAG_DROP_TABLE
drop-view-statement	"DROP VIEW"	SQL_DIAG_DROP_VIEW
grant-statement	"GRANT"	SQL_DIAG_GRANT
insert-statement	"INSERT"	SQL_DIAG_INSERT
ODBC-procedure-extension	"CALL"	SQL_DIAG_PROCEDURE_CALL
revoke-statement	"REVOKE"	SQL_DIAG_REVOKE
update-statement-positioned	"DYNAMIC UPDATE CURSOR"	SQL_DIAG_DYNAMIC_UPDATE_ CURSOR
update-statement-searched	"UPDATE WHERE"	SQL_DIAG_UPDATE_WHERE
merge-statement	"MERGE"	SQL_DIAG_MERGE
Unknown	empty string	SQL_DIAG_UNKNOWN_ STATEMENT

Related concepts:

- "Deferred prepare in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQLGetDiagField function (CLI) - Get a field of diagnostic data" on page 168

Chapter 5. Data type attributes

This chapter describes the following attributes for SQL data types supported by DB2 CLI:

- precision
- scale
- length
- display size

Data type precision (CLI) table

The precision of a numeric column or parameter refers to the maximum number of digits used by the data type of the column or parameter. The precision of a non-numeric column or parameter generally refers to the maximum or the defined number of characters of the column or parameter. The following table defines the precision for each SQL data type.

Table 160. Precision

fSqlType	Precision
SQL_CHAR SQL_VARCHAR SQL_CLOB	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR	The maximum length of the column or parameter. [a]
SQL_DECIMAL SQL_DECFLOAT SQL_NUMERIC	The defined maximum number of digits. For example, the precision of a column defined as NUMERIC(10,3) is 10 and the precision of a column defined as DECFLOAT(34) is 34.
SQL_SMALLINT [b]	5
SQL_BIGINT	19
SQL_INTEGER [b]	10
SQL_FLOAT [b]	15
SQL_REAL [b]	7
SQL_DOUBLE [b]	15
SQL_BINARY SQL_VARBINARY SQL_BLOB	The defined length of the column or parameter. For example, the precision of a column defined as CHAR(10) FOR BIT DATA, is 10.
SQL_LONGVARBINARY	The maximum length of the column or parameter.
SQL_DATE [b]	10 (the number of characters in the yyyy-mm-dd format).
SQL_TIME [b]	8 (the number of characters in the hh:mm:ss format).

Table 160. Precision (continued)

fSqlType	Precision
SQL_TIMESTAMP	The number of characters in the "yyy-mm-dd hh:mm:ss[.fff[fff]]" format used by the TIMESTAMP data type. For example, if a timestamp does not use seconds or fractional seconds, the precision is 16 (the number of characters in the "yyyy-mm-dd hh:mm" format). If a timestamp uses thousandths of a second, the precision is 23 (the number of characters in the "yyyy-mm-dd hh:mm:ss.fff" format).
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	The defined length of the column or parameter. For example, the precision of a column defined as GRAPHIC(10) is 10.
SQL_LONGVARGRAPHIC	The maximum length of the column or parameter.
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	The defined length of the column or parameter. For example, the precision of a column defined as WCHAR(10) is 10.
SQL_XML	0, unless the XML value is an argument to external routines. For external routines, the precision is the defined length, n, of an XML AS CLOB(n) argument.
Note: [a] When defining the precision of a parameter of this data type with `SQLBindParameter()` or `SQLSetParam()`, *cbParamDef* should be set to the total length of the data, not the precision as defined in this table. [b] The *cbColDef* argument of `SQLBindParameter()` is ignored for this data type.	

Related concepts:

- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Data type scale (CLI) table

The scale of a numeric column or parameter refers to the maximum number of digits to the right of the decimal point. Note that, for approximate floating point number columns or parameters, the scale is undefined, since the number of digits to the right of the decimal place is not fixed. The following table defines the scale for each SQL data type.

Table 161. Scale

fSqlType	Scale
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Not applicable.
SQL_DECIMAL SQL_NUMERIC	The defined number of digits to the right of the decimal place. For example, the scale of a column defined as NUMERIC(10, 3) is 3.

Table 161. Scale (continued)

fSqlType	Scale
SQL_SMALLINT SQL_INTEGER SQL_BIGINT	0
SQL_REAL SQL_FLOAT SQL_DECFLOAT SQL_DOUBLE	Not applicable.
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	Not applicable.
SQL_DATE SQL_TIME	Not applicable.
SQL_TIMESTAMP	The number of digits to the right of the decimal point in the "yyyy-mm-dd hh:mm:ss[fff[fff]]" format. For example, if the TIMESTAMP data type uses the "yyyy-mm-dd hh:mm:ss.fff" format, the scale is 3.
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLOB	Not applicable.
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	Not applicable.
SQL_XML	Not applicable.

Related concepts:

- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Data type length (CLI) table

The length of a column is the maximum number of *bytes* returned to the application when data is transferred to its default C data type. For character data, the length does not include the null termination byte. Note that the length of a column might be different than the number of bytes required to store the data on the data source.

The following table defines the length for each SQL data type.

Table 162. Length

fSqlType	Length
SQL_CHAR SQL_VARCHAR SQL_CLOB	The defined length of the column. For example, the length of a column defined as CHAR(10) is 10.

Table 162. Length (continued)

fSqlType	Length
SQL_LONGVARCHAR	The maximum length of the column.
SQL_DECIMAL SQL_NUMERIC	The maximum number of digits plus two. Since these data types are returned as character strings, characters are needed for the digits, a sign, and a decimal point. For example, the length of a column defined as NUMERIC(10,3) is 12.
SQL_DECFLOAT	If the column is defined as DECFLOAT(16) then the length is 8. If the column is defined as DECFLOAT(34) then the length is 16.
SQL_SMALLINT	2 (two bytes).
SQL_INTEGER	4 (four bytes).
SQL_BIGINT	8 (eight bytes).
SQL_REAL	4 (four bytes).
SQL_FLOAT	8 (eight bytes).
SQL_DOUBLE	8 (eight bytes).
SQL_BINARY SQL_VARBINARY SQL_BLOB	The defined length of the column. For example, the length of a column defined as CHAR(10) FOR BIT DATA is 10.
SQL_LONGVARBINARY	The maximum length of the column.
SQL_DATE SQL_TIME	6 (the size of the DATE_STRUCT or TIME_STRUCT structure).
SQL_TIMESTAMP	16 (the size of the TIMESTAMP_STRUCT structure).
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	The defined length of the column times 2. For example, the length of a column defined as GRAPHIC(10) is 20.
SQL_LONGVARGRAPHIC	The maximum length of the column times 2.
SQL_WCHAR SQL_WVARCHAR SQL_WLONGVARCHAR	The defined length of the column times 2. For example, the length of a column defined as WCHAR(10) is 20.
SQL_XML	0 (stored XML documents are limited to 2GB in size however)

Related concepts:

- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Data type display (CLI) table

The display size of a column is the maximum number of *bytes* needed to display data in character form. The following table defines the display size for each SQL data type.

Table 163. Display size

fSqlType	Display size
SQL_CHAR SQL_VARCHAR SQL_CLOB	The defined length of the column. For example, the display size of a column defined as CHAR(10) is 10.
SQL_LONGVARCHAR	The maximum length of the column.
SQL_DECIMAL SQL_NUMERIC	The precision of the column plus two (a sign, precision digits, and a decimal point). For example, the display size of a column defined as NUMERIC(10,3) is 12.
SQL_DECFLOAT	If the column is defined as DECFLOAT(16) then the display length is 23. If the column is defined as DECFLOAT(34) then the display length is 42.
SQL_SMALLINT	6 (a sign and 5 digits).
SQL_INTEGER	11 (a sign and 10 digits).
SQL_BIGINT	20 (a sign and 19 digits).
SQL_REAL	13 (a sign, 7 digits, a decimal point, the letter E, a sign, and 2 digits).
SQL_FLOAT SQL_DOUBLE	22 (a sign, 15 digits, a decimal point, the letter E, a sign, and 3 digits).
SQL_BINARY SQL_VARBINARY SQL_BLOB	The defined maximum length of the column times 2 (each binary byte is represented by a 2 digit hexadecimal number). For example, the display size of a column defined as CHAR(10) FOR BIT DATA is 20.
SQL_LONGVARBINARY	The maximum length of the column times 2.
SQL_DATE	10 (a date in the format yyyy-mm-dd).
SQL_TIME	8 (a time in the format hh:mm:ss).
SQL_TIMESTAMP	19 (if the scale of the timestamp is 0) or 20 plus the scale of the timestamp (if the scale is greater than 0). This is the number of characters in the "yyyy-mm-dd hh:mm:ss[fff[fff]]" format. For example, the display size of a column storing thousandths of a second is 23 (the number of characters in "yyyy-mm-dd hh:mm:ss.fff").
SQL_GRAPHIC SQL_VARGRAPHIC SQL_DBCLOB	Twice the defined length of the column or parameter. For example, the display size of a column defined as GRAPHIC(10) is 20.
SQL_LONGVARGRAPHIC	The maximum length of the column or parameter.
SQL_XML	0

Related concepts:

- "Data types and data conversion in CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Related reference:

- "SQL symbolic and default data types for CLI applications" in *Call Level Interface Guide and Reference, Volume 1*

Appendix A. 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 401

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 396

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 164. 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 164. 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 165. DB2 Connect-specific technical information

Name	Form Number	Available in print
DB2 Connect User's Guide	SC10-4229	Yes

Table 165. 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 166. 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 395
- "About the Release Notes" in *Release notes*

Related tasks:

- "Ordering printed DB2 books" on page 398

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 395

Related reference:

- "DB2 technical library in hardcopy or PDF format" on page 396

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 401

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 395

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 395

Terms and Conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal use: You may reproduce these Publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these Publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these Publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these Publications, or reproduce, distribute or display these Publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the Publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the Publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED ″AS-IS″ AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Appendix B. 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information may contain sample application programs, in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _*enter the year or years*_. All rights reserved.

Trademarks

Company, product, or service names identified in the documents of the DB2 Version 9 documentation library may be trademarks or service marks of International Business Machines Corporation or other companies. Information on the trademarks of IBM Corporation in the United States, other countries, or both is located at http://www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 documentation library:

Microsoft, Windows, Windows NT®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel®, Itanium®, Pentium®, and Xeon® are trademarks of Intel Corporation in the United States, other countries, or both.

Java™ and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Appendix C. Further notices for the DB2 Call Level Interface Guide and Reference

This book incorporates text which is copyright The X/Open Company Limited. The text was taken by permission from:

X/Open CAE Specification, March 1995,
Data Management: SQL Call Level Interface (CLI)
(ISBN: 1-85912-081-4, C451).

X/Open Preliminary Specification, March 1995,
Data Management: Structured Query Language (SQL), Version 2
(ISBN: 1-85912-093-8, P446).

This book incorporates text which is copyright 1992, 1993, 1994, 1997 by Microsoft Corporation. The text was taken by permission from Microsoft's *ODBC 2.0 Programmer's Reference and SDK Guide* ISBN 1-55615-658-8, and from Microsoft's *ODBC 3.0 Software Development Kit and Programmer's Reference* ISBN 1-57231-516-4.

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

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

SC10-4225-00

Spine information:

IBM DB2 **DB2 Version 9** **CLI Guide and Reference, Volume 2**