



IBM Systems - iSeries

Database

DB2 UDB SQL Call Level Interface (ODBC)

*Version 5 Release 4*







IBM Systems - iSeries

Database

DB2 UDB SQL Call Level Interface (ODBC)

*Version 5 Release 4*

**Note**

Before using this information and the product it supports, read the information in "Notices," on page 261.

**Eighth Edition (February 2006)**

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 1999, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

## SQL call level interface . . . . . 1

What's new for V5R4 . . . . .	1
Printable PDF . . . . .	2
Get started with DB2 UDB CLI . . . . .	2
Differences between DB2 UDB CLI and embedded SQL . . . . .	3
Advantages of using DB2 UDB CLI instead of embedded SQL . . . . .	5
Decide between DB2 UDB CLI, dynamic SQL, and static SQL . . . . .	5
Write a DB2 UDB CLI application . . . . .	5
Initialization and termination tasks in a DB2 UDB CLI application . . . . .	6
Transaction processing task in a DB2 UDB CLI application . . . . .	9
Diagnostics in a DB2 UDB CLI application . . . . .	14
Data types and data conversion in DB2 UDB CLI functions . . . . .	15
Work with string arguments in DB2 UDB CLI functions . . . . .	17
DB2 UDB CLI functions . . . . .	18
Categories of DB2 UDB CLIs . . . . .	19
SQLAllocConnect - Allocate connection handle . . . . .	21
SQLAllocEnv - Allocate environment handle . . . . .	24
SQLAllocHandle - Allocate handle. . . . .	26
SQLAllocStmt - Allocate a statement handle . . . . .	27
SQLBindCol - Bind a column to an application variable. . . . .	29
SQLBindFileToCol - Bind LOB file reference to LOB column . . . . .	33
SQLBindFileToParam - Bind LOB file reference to LOB parameter . . . . .	36
SQLBindParam - Bind a buffer to a parameter marker . . . . .	38
SQLBindParameter - Bind a parameter marker to a buffer. . . . .	42
SQLCancel - Cancel statement . . . . .	50
SQLCloseCursor - Close cursor statement . . . . .	51
SQLColAttributes - Obtain column attributes . . . . .	52
SQLColumnPrivileges - Get privileges associated with the columns of a table . . . . .	56
SQLColumns - Get column information for a table. . . . .	58
SQLConnect - Connect to a data source . . . . .	62
SQLCopyDesc - Copy description statement . . . . .	64
SQLDataSources - Get list of data sources . . . . .	65
SQLDescribeCol - Describe column attributes . . . . .	68
SQLDescribeParam - Return description of a parameter marker . . . . .	70
SQLDisconnect - Disconnect from a data source . . . . .	73
SQLDriverConnect - (Expanded) Connect to a data source . . . . .	74
SQLEndTran - Commit or roll back a transaction . . . . .	77
SQLError - Retrieve error information . . . . .	79
SQLExecDirect - Execute a statement directly . . . . .	81
SQLExecute - Execute a statement. . . . .	83

SQLExtendedFetch - Fetch array of rows . . . . .	85
SQLFetch - Fetch next row . . . . .	87
SQLFetchScroll - Fetch from a scrollable cursor . . . . .	92
SQLForeignKeys - Get the list of foreign key columns . . . . .	94
SQLFreeConnect - Free connection handle . . . . .	99
SQLFreeEnv - Free environment handle . . . . .	100
SQLFreeHandle - Free a handle . . . . .	101
SQLFreeStmt - Free (or reset) a statement handle . . . . .	102
SQLGetCol - Retrieve one column of a row of the result set . . . . .	103
SQLGetConnectAttr - Get the value of a connection attribute . . . . .	109
SQLGetConnectOption - Return current setting of a connect Option . . . . .	110
SQLGetCursorName - Get cursor name. . . . .	111
SQLGetData - Get data from a column . . . . .	114
SQLGetDescField - Get descriptor field. . . . .	115
SQLGetDescRec - Get descriptor record. . . . .	117
SQLGetDiagField - Return diagnostic information (extensible) . . . . .	119
SQLGetDiagRec - Return diagnostic information (concise) . . . . .	121
SQLGetEnvAttr - Return current setting of an environment Attribute . . . . .	123
SQLGetFunctions - Get functions. . . . .	124
SQLGetInfo - Get general information . . . . .	127
SQLGetLength - Retrieve length of a string value . . . . .	140
SQLGetPosition - Return starting position of string . . . . .	141
SQLGetStmtAttr - Get the value of a statement attribute . . . . .	143
SQLGetStmtOption - Return current setting of a statement option . . . . .	145
SQLGetSubString - Retrieve portion of a string value . . . . .	146
SQLGetTypeInfo - Get data type information . . . . .	149
SQLLanguages - Get SQL dialect or conformance information . . . . .	153
SQLMoreResults - Determine whether there are more result sets . . . . .	155
SQLNativeSql - Get native SQL text . . . . .	156
SQLNextResult - Process the next result set . . . . .	158
SQLNumParams - Get number of parameters in an SQL statement . . . . .	159
SQLNumResultCols - Get number of result columns . . . . .	160
SQLParamData - Get next parameter for which a data value is needed . . . . .	162
SQLParamOptions - Specify an input array for a parameter . . . . .	163
SQLPrepare - Prepare a statement . . . . .	164
SQLPrimaryKeys - Get primary key columns of a table. . . . .	168

SQLProcedureColumns - Get input/output parameter information for a procedure . . . . .	170
SQLProcedures - Get list of procedure names . . . . .	176
SQLPutData - Pass data value for a parameter . . . . .	179
SQLReleaseEnv - Release all environment resources . . . . .	181
SQLRowCount - Get row count . . . . .	182
SQLSetConnectAttr - Set a connection attribute . . . . .	183
SQLSetConnectOption - Set connection option . . . . .	190
SQLSetCursorName - Set cursor name . . . . .	192
SQLSetDescField - Set a descriptor field . . . . .	193
SQLSetDescRec - Set a descriptor record . . . . .	195
SQLSetEnvAttr - Set environment attribute . . . . .	196
SQLSetParam - Set parameter . . . . .	200
SQLSetStmtAttr - Set a statement attribute. . . . .	201
SQLSetStmtOption - Set statement option . . . . .	205
SQLSpecialColumns - Get special (row identifier) columns . . . . .	206
SQLStatistics - Get index and statistics information for a base table . . . . .	209
SQLTablePrivileges - Get privileges associated with a table . . . . .	212

SQLTables - Get table information . . . . .	215
SQLTransact - Commit or roll back transaction . . . . .	217
DB2 UDB CLI include file . . . . .	219
Run DB2 UDB CLI in server mode . . . . .	246
Start DB2 UDB CLI in SQL server mode . . . . .	246
Restrictions for running DB2 UDB CLI in server mode . . . . .	247
Examples: DB2 UDB CLI applications . . . . .	247
Example: Embedded SQL and the equivalent DB2 UDB CLI function calls . . . . .	247
Example: Use the CLI XA transaction connection attributes . . . . .	250
Example: Interactive SQL and the equivalent DB2 UDB CLI function calls . . . . .	253
Code license and disclaimer information . . . . .	260

<b>Appendix. Notices . . . . .</b>	<b>261</b>
Programming Interface Information . . . . .	262
Trademarks . . . . .	263
Terms and conditions. . . . .	263

---

## SQL call level interface

DB2<sup>®</sup> UDB call level interface (CLI) is a callable Structured Query Language (SQL) programming interface that is supported in all DB2 environments except for DB2 Universal Database for z/OS<sup>®</sup> and DB2 Server for VSE and VM.

A *callable SQL interface* is a WinSock application programming interface (API) for database access that uses function calls to start dynamic SQL statements.

DB2 UDB CLI is an alternative to embedded dynamic SQL. The important difference between embedded dynamic SQL and DB2 UDB CLI is how the SQL statements are started. On the iSeries<sup>™</sup> server, this interface is available to any of the ILE languages.

DB2 UDB CLI also provides full Level 1 Microsoft<sup>®</sup> Open Database Connectivity (ODBC) support, plus many Level 2 functions. For the most part, ODBC is a superset of the ANS and ISO SQL CLI standard.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

---

## What’s new for V5R4

This topic highlights numerous changes to the SQL call level interface for V5R4.

- | The limit for the total number of concurrently allocated handles is expanded from 80 000 to 160 000.
- | New environment, connection and statement attributes are added, including:
  - | • Cursor sensitivity statement attribute
  - | • New cursor type statement attribute (SQL\_CURSOR\_STATIC)
  - | • New query optimizer connection attribute(SQL\_ATTR\_QUERY\_OPTIMIZE\_GOAL)
- | New SQLGetInfo and SQLColAttributes options are added, including:
  - | • User name for a connection from SQLGetInfo(): SQL\_USER\_NAME
  - | • Database name for a connection from SQLGetInfo(): SQL\_DATABASE\_NAME
  - | • Display the size needed to display a data type from SQLColAttributes(): SQL\_DESC\_DISPLAY\_SIZE
- | New supports are added, including:
  - | • XA support through the CLI connection attributes SQL\_ATTR\_TXN\_EXTERNAL and SQL\_ATTR\_TXN\_INFO
  - | • Support for array (block) fetching and column-wise binding in the SQLFetchScroll()
  - | • 2-megabyte SQL statement support through the CLI interface

**Note:** This is not a complete list of the new supports.



The following APIs are changed in this release:

- | • “SQLConnect - Connect to a data source” on page 62
- | • “SQLFetchScroll - Fetch from a scrollable cursor” on page 92
- | • “SQLGetConnectOption - Return current setting of a connect Option” on page 110
- | • “SQLGetDescField - Get descriptor field” on page 115
- | • “SQLGetDescRec - Get descriptor record” on page 117

- | • “SQLGetInfo - Get general information” on page 127
- | • “SQLGetStmtOption - Return current setting of a statement option” on page 145
- | • “SQLGetTypeInfo - Get data type information” on page 149
- | • “SQLSetConnectAttr - Set a connection attribute” on page 183
- | • “SQLSetConnectOption - Set connection option” on page 190
- | • “SQLSetEnvAttr - Set environment attribute” on page 196
- | • “SQLSetStmtAttr - Set a statement attribute” on page 201
- | • “SQLSetStmtOption - Set statement option” on page 205

## How to see what's new or changed

To help you see where technical changes have been made, this information uses:

- The  image to mark where new or changed information begins.
- The  image to mark where new or changed information ends.

To find other information about what's new or changed this release, see the Memo to users.

---

## Printable PDF

Use this to view and print a PDF of this information.


To view or download the PDF version of this document, select SQL call level interface (about 2650 KB).

## Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF in your browser (right-click the link above).
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

## Downloading Adobe Reader

- | You need Adobe Reader installed on your system to view or print these PDFs. You can download a free
- | copy from the Adobe Web site ([www.adobe.com/products/acrobat/readstep.html](http://www.adobe.com/products/acrobat/readstep.html))  .

---

## Get started with DB2 UDB CLI

Learn about the basics of the DB2 UDB CLI, how it compares to embedded SQL, and how to select the best interface for your programming needs.

It is important to understand what DB2 UDB CLI, or any callable SQL interface, is based on, and compare it with existing interfaces.

ISO standard 9075:1999 – Database Language SQL Part 3: Call-Level Interface provides the standard definition of CLI. The goal of this interface is to increase the portability of applications by enabling them to become independent of any one database server.

ODBC provides a Driver Manager for Windows<sup>®</sup>, which offers a central point of control for each ODBC driver (a dynamic link library (DLL) that implements ODBC function calls and interacts with a specific Database Management System (DBMS)).



## Where to find answers to additional DB2 UDB CLI questions

An FAQ, which elaborates on some items discussed in this topic collection, is available on the IBM DB2 Universal Database Web site.

## Differences between DB2 UDB CLI and embedded SQL

An application that uses an embedded SQL interface requires a precompiler to convert the SQL statements into code. Code is compiled, bound to the database, and processed. In contrast, a DB2 UDB CLI application does not require precompilation or binding, but instead uses a standard set of functions to run SQL statements and related services at run time.

This difference is important because, traditionally, precompilers have been specific to a database product, which effectively ties your applications to that product. DB2 UDB CLI enables you to write portable applications that are independent of any particular database product. This independence means that a DB2 UDB CLI application does not need to be recompiled or rebound to access-different database products. An application selects the appropriate database products at run time.

DB2 UDB CLI and embedded SQL also differ in the following ways:

- DB2 UDB CLI does not require the explicit declaration of cursors. DB2 UDB CLI generates them as needed. The application can then use the generated cursor in the normal cursor fetch model for multiple row SELECT statements and positioned UPDATE and DELETE statements.
- The OPEN statement is not necessary in DB2 UDB CLI. Instead, the processing of a SELECT automatically causes a cursor to be opened.
- Unlike embedded SQL, DB2 UDB CLI allows the use of parameter markers on the equivalent of the EXECUTE IMMEDIATE statement (the SQLExecDirect() function).
- A COMMIT or ROLLBACK in DB2 UDB CLI is issued through the SQLTransact() or SQLEndTran() function call rather than by passing it as an SQL statement.
- DB2 UDB CLI manages statement-related information on behalf of the application, and provides a *statement handle* to refer to it as an abstract object. This handle avoids the need for the application to use product-specific data structures.
- Similar to the statement handle, the *environment handle* and *connection handle* provide a means to refer to all global variables and connection specific information.
- DB2 UDB CLI uses the SQLSTATE values defined by the X/Open SQL CAE specification. Although the format and many of the values are consistent with values that are used by the IBM® relational database products, there are differences.

Despite these differences, there is an important common concept between embedded SQL and DB2 UDB CLI:

- DB2 UDB CLI can process any SQL statement that can be prepared dynamically in embedded SQL. This is guaranteed because DB2 UDB CLI does not actually process the SQL statement itself, but passes it to the Database Management System (DBMS) for dynamic processing.

Table 1 lists each SQL statement, and whether it can be processed using DB2 UDB CLI.

Table 1. SQL statements

SQL statement	Dyn <sup>1</sup>	CLI <sup>3</sup>
ALTER TABLE	X	X
BEGIN DECLARE SECTION <sup>2</sup>		
CALL	X	X
CLOSE		SQLFreeStmt()
COMMENT ON	X	X

Table 1. SQL statements (continued)

SQL statement	Dyn <sup>1</sup>	CLI <sup>3</sup>
COMMIT	X	SQLTransact(), SQLEndTran()
CONNECT (Type 1)		SQLConnect()
CONNECT (Type 2)		SQLConnect()
CREATE INDEX	X	X
CREATE TABLE	X	X
CREATE VIEW	X	X
DECLARE CURSOR <sup>b</sup>		SQLAllocStmt()
DELETE	X	X
DESCRIBE		SQLDescribeCol(), SQLColAttributes()
DISCONNECT		SQLDisconnect()
DROP	X	X
END DECLARE SECTION <sup>b</sup>		
EXECUTE		SQLExecute()
EXECUTE IMMEDIATE		SQLExecDirect()
FETCH		SQLFetch()
GRANT	X	X
INCLUDE <sup>b</sup>		
INSERT	X	X
LOCK TABLE	X	X
OPEN		SQLExecute(), SQLExecDirect()
PREPARE		SQLPrepare()
RELEASE		SQLDisconnect()
REVOKE	X	X
ROLLBACK	X	SQLTransact(), SQLEndTran()
SELECT	X	X
SET CONNECTION		
UPDATE	X	X
WHENEVER <sup>2</sup>		

**Notes:**

<sup>1</sup> *Dyn* stands for dynamic. All statements in this list can be coded as static SQL, but only those marked with X can be coded as dynamic SQL.

<sup>2</sup> This is a nonprocessable statement.

<sup>3</sup> An X indicates that this statement can be processed using either SQLExecDirect() or SQLPrepare() and SQLExecute(). If there is an equivalent DB2 UDB CLI function, the function name is listed.

Each DBMS might have additional statements that can be dynamically prepared, in which case DB2 UDB CLI passes them to the DBMS. There is one exception, COMMIT and ROLLBACK can be dynamically prepared by some DBMSs but are not passed. Instead, the SQLTransact() or SQLEndTran() should be used to specify either COMMIT or ROLLBACK.

## Advantages of using DB2 UDB CLI instead of embedded SQL

The DB2 UDB CLI interface has several key advantages over embedded SQL.

- It is ideally suited for a client-server environment, in which the target database is not known when the application is built. It provides a consistent interface for executing SQL statements, regardless of which database server to which the application is connected .
- It increases the portability of applications by removing the dependence on precompilers. Applications are distributed not as compiled applications or runtime libraries but as source code which are preprocessed for each database product.
- DB2 UDB CLI applications do not need to be bound to each database to which they connect.
- DB2 UDB CLI applications can connect to multiple databases simultaneously.
- DB2 UDB CLI applications are not responsible for controlling global data areas, such as SQL Communication Area (SQLCA) and SQL Descriptor Area (SQLDA), as they are with embedded SQL applications. Instead, DB2 UDB CLI allocates and controls the necessary data structures, and provides a *handle* for the application to refer to them.

## Decide between DB2 UDB CLI, dynamic SQL, and static SQL

Which interfaces you choose depend on your application.

DB2 UDB CLI is ideally suited for query-based applications requiring portability, and not requiring the APIs or utilities offered by a particular Database Management System (DBMS) (for example, catalog database, backup, restore). This does not mean that using DB2 UDB CLI calls DBMS specific APIs from an application. It means that the application is no longer as portable.

Another important consideration is the performance comparison between dynamic and static SQL. Dynamic SQL is prepared at run time, while static SQL is prepared at the precompile stage. Because preparing statements requires additional processing time, static SQL might be more efficient. If you choose static over dynamic SQL, then DB2 UDB CLI is not an option.

In most cases the choice between either interface is open to personal preference. Your previous experience might make one alternative seem more intuitive than the other.

---

## Write a DB2 UDB CLI application

Find out how you code your applications to use the DB2 CLI functions.

A DB2 UDB CLI application consists of a set of tasks, each comprised of a set of discrete steps. Other tasks might occur throughout the application as it runs. The application calls one or more DB2 UDB CLI functions to carry out each of these tasks.

Every DB2 UDB CLI application contains the three main tasks that are shown in the following figure. If the functions are not called in the sequence that is shown in the figure, an error results.

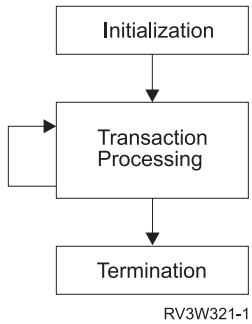


Figure 1. Conceptual view of a DB2 UDB CLI application

The *initialization* task allocates and initializes resources in preparation for the main *Transaction Processing* task.

The *transaction processing* task, the main task of the application, passes queries and modifications to the SQL to DB2 UDB CLI.

The *termination* task frees allocated resources. The resources generally consist of data areas that are identified by unique handles. After freeing the resources, other tasks can use these handles.

In addition to the three central tasks that control a DB2 UDB CLI application, there are numerous *general* tasks, such as diagnostic message handlers, throughout an application.

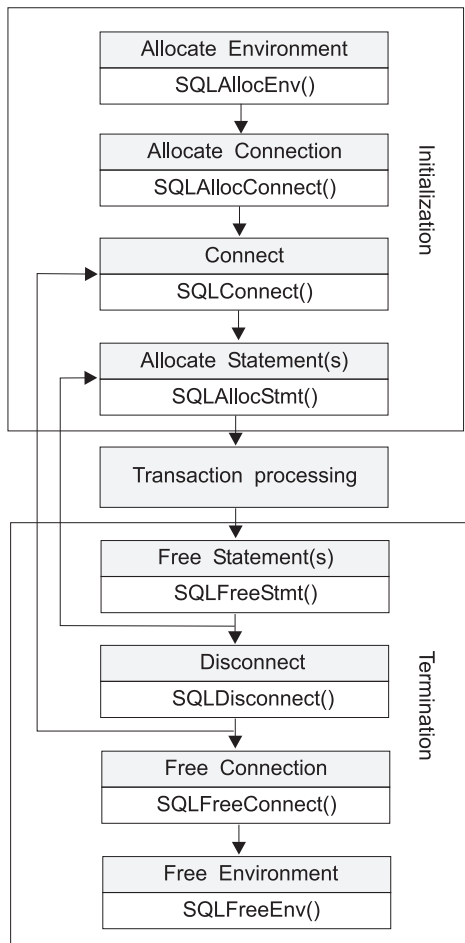
See “Categories of DB2 UDB CLIs” on page 19 for an overview of how the CLI functions fit into these key task areas.

This topic provides examples to illustrate how these functions are used in a DB2 UDB CLI application.

## Initialization and termination tasks in a DB2 UDB CLI application

The initialization task allocates and initializes environment handles and connection handles.

The following figure shows the function call sequences for both the initialization and termination tasks. The transaction processing task in the middle of the diagram is shown in “Transaction processing task in a DB2 UDB CLI application” on page 9.



RV3W322-1

Figure 2. Conceptual view of initialization and termination tasks

The termination task frees handles. A handle is a variable that refers to a data object that is controlled by DB2 UDB CLI. Using handles frees the application from having to allocate and manage global variables or data structures, such as the SQL descriptor area (SQLDA), or SQL communication area (SQLCA) used in embedded SQL interfaces for IBM Database Management Systems (DBMSs). An application then passes the appropriate handle when it calls other DB2 UDB CLI functions. Here are the types of handles:

### Environment handle

The environment handle refers to the data object that contains global information regarding the state of the application. This handle is allocated by calling `SQLAllocEnv()`, and freed by calling `SQLFreeEnv()`. An environment handle must be allocated before a connection handle can be allocated. Only one environment handle can be allocated per application.

### Connection handle

A connection handle refers to a data object that contains information that is associated with a connection that is managed by DB2 UDB CLI. This includes general status information, transaction status, and diagnostic information. Each connection handle is allocated by calling `SQLAllocConnect()` and freed by calling `SQLFreeConnect()`. An application must allocate a connection handle for each connection to a database server.

### Statement handle

Statement handles are discussed in Transaction processing task in a DB2 UDB CLI application.

## Example: Initialization and connection in a DB2 UDB CLI application

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```
/******  
** file = basiccon.c  
**   - demonstrate basic connection to two datasources.  
**   - error handling ignored for simplicity  
**  
** Functions used:  
**  
**   SQLAllocConnect  SQLDisconnect  
**   SQLAllocEnv     SQLFreeConnect  
**   SQLConnect      SQLFreeEnv  
**  
**  
*****/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include "sqlcli.h"  
  
int  
connect(SQLHENV henv,  
        SQLHDBC * hdbc);  
  
#define MAX_DSN_LENGTH  18  
#define MAX_UID_LENGTH  10  
#define MAX_PWD_LENGTH  10  
#define MAX_CONNECTIONS 5  
  
int  
main()  
{  
    SQLHENV     henv;  
    SQLHDBC     hdbc[MAX_CONNECTIONS];  
  
    /* allocate an environment handle */  
    SQLAllocEnv(&henv);  
  
    /* Connect to first data source */  
    connect(henv, &hdbc[0]);  
  
    /* Connect to second data source */  
    connect(henv, &hdbc[1]);  
  
    /****** Start Processing Step ******/  
    /* allocate statement handle, execute statement, and so forth */  
    /****** End Processing Step ******/  
  
    printf("\nDisconnecting ....\n");  
    SQLDisconnect(hdbc[0]); /* disconnect first connection */  
    SQLDisconnect(hdbc[1]); /* disconnect second connection */  
    SQLFreeConnect(hdbc[0]); /* free first connection handle */  
    SQLFreeConnect(hdbc[1]); /* free second connection handle */  
    SQLFreeEnv(henv); /* free environment handle */  
  
    return (SQL_SUCCESS);  
}  
  
/******  
** connect - Prompt for connect options and connect **  
*****/  
  
int  
connect(SQLHENV henv,
```

```

        SQLHDBC * hdbc)
{
    SQLRETURN      rc;
    SQLCHAR        server[MAX_DSN_LENGTH + 1], uid[MAX_UID_LENGTH + 1],
pwd[MAX_PWD_LENGTH
+ 1];
    SQLCHAR        buffer[255];
    SQLSMALLINT    outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    SQLAllocConnect(henv, hdbc); /* allocate a connection handle */

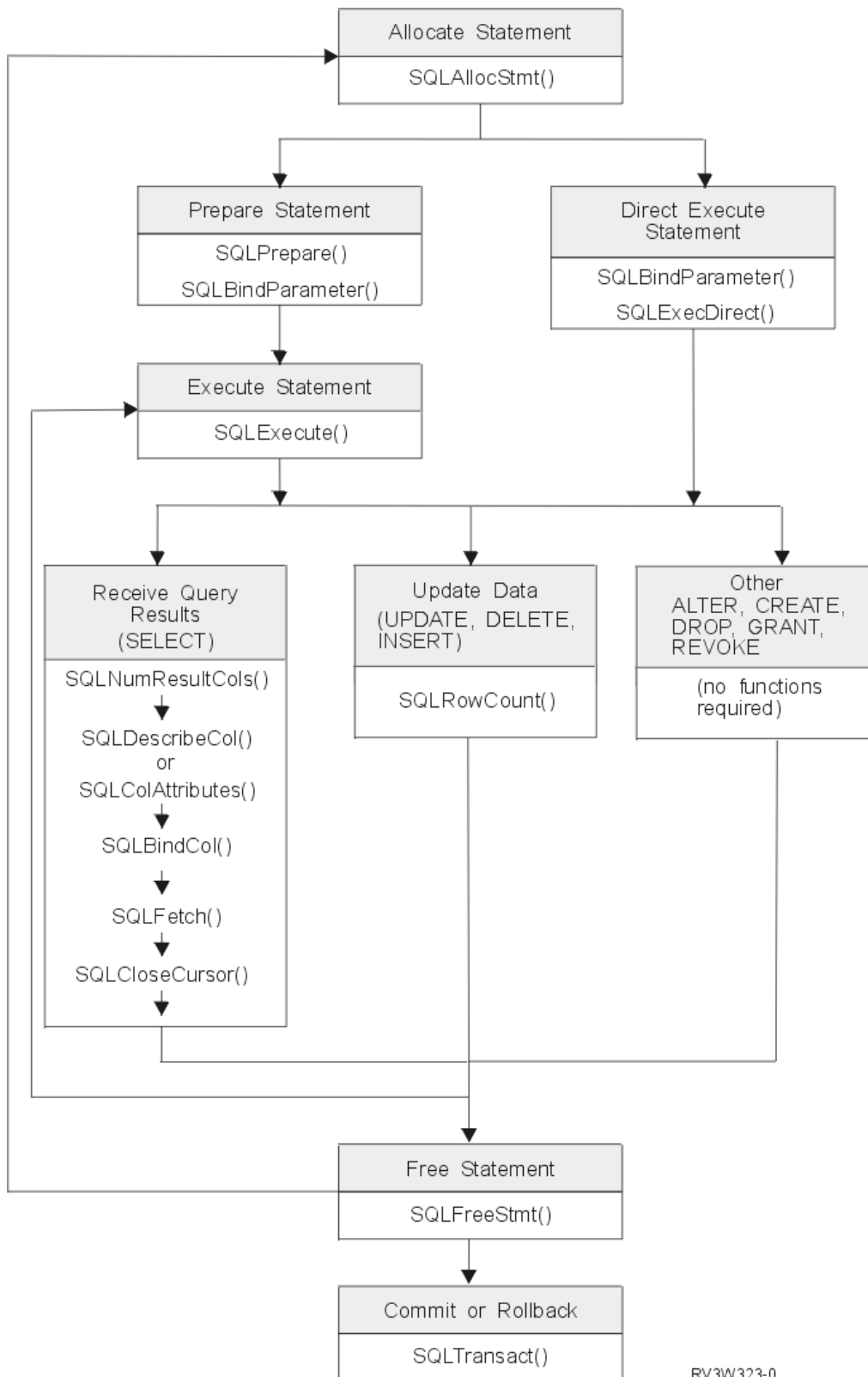
    rc = SQLConnect(*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database\n");
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
        return (SQL_SUCCESS);
    }
}

```

## Transaction processing task in a DB2 UDB CLI application

This topic shows a typical order of function calls in a DB2 UDB CLI application.

The typical order of function calls in a DB2 UDB CLI application is shown in the following figure. The figure does not show all functions or possible paths.



RV3W323-0

Figure 3. Transaction processing

The figure shows the steps and the DB2 UDB CLI functions in the transaction processing task. This task contains these steps:

1. "Allocate statement handle(s) in a DB2 UDB CLI application" on page 11



2. "Preparation and processing tasks in a DB2 UDB CLI application"
3. "Processing results in a DB2 UDB CLI application" on page 12
4. "Free statement handles in a DB2 UDB CLI application" on page 14
5. "Commit or roll back in a DB2 UDB CLI application" on page 14

The function `SQLAllocStmt` is needed to obtain a statement handle that is used to process the SQL statement. There are two methods of statement processing that can be used. By using `SQLPrepare` and `SQLExecute`, the program can break the process into two steps. The function `SQLBindParameter` is used to bind program addresses to host variables used in the prepared SQL statement. The second method is the direct processing method in which `SQLPrepare` and `SQLExecute` are replaced by a single call to `SQLExecDirect`.

As soon as the statement is processed, the remaining processing depends on the type of SQL statement. For `SELECT` statements, the program uses functions like `SQLNumResultCols`, `SQLDescribeCol`, `SQLBindCol`, `SQLFetch`, and `SQLCloseCursor` to process the result set. For statements that update data, `SQLRowCount` can be used to determine the number of affected rows. For other types of SQL statements, the processing is complete after the statement is processed. `SQLFreeStmt` is then used in all cases to indicate that the handle is no longer needed.

### Allocate statement handle(s) in a DB2 UDB CLI application

`SQLAllocStmt()` allocates a statement handle. A statement handle refers to the data object that contains information about an SQL statement that is managed by DB2 UDB CLI. This includes information such as dynamic arguments, cursor information, bindings for dynamic arguments and columns, result values, and status information (these are discussed later). Each statement handle is associated with a connection handle.

- | Allocate a statement handle in order to run a statement. You can concurrently allocate up to 160 000
- | handles. This applies to all types of handles, including descriptor handles that are implicitly allocated by
- | the implementation code.

### Preparation and processing tasks in a DB2 UDB CLI application

As soon as a statement handle has been allocated, there are two methods of specifying and executing SQL statements:

1. Prepare, and then execute:
  - a. Call `SQLPrepare()` with an SQL statement as an argument.
  - b. Call `SQLSetParam()`, if the SQL statement contains *parameter markers*.
  - c. Call `SQLExecute()`
2. Execute direct:
  - a. Call `SQLSetParam()`, if the SQL statement contains *parameter markers*.
  - b. Call `SQLExecDirect()` with an SQL statement as an argument.

The first method splits the preparation of the statement from the processing. This method is used when:

- The statement is processed repeatedly (typically with different parameter values). This avoids having to prepare the same statement more than once.
- The application requires information about the columns in the result set, before statement processing.

The second method combines the preparation step and the processing step into one. This method is used when:

- The statement is processed once. This avoids having to call two functions to process the statement.
- The application does not require information about the columns in the result set, before the statement is processed.

### Binding parameters in SQL statements in a DB2 UDB CLI application

Both processing methods allow the use of parameter markers in place of an *expression* (or host variable in embedded SQL) in an SQL statement.

Parameter markers are represented by the '?' character and indicate the position in the SQL statement where the contents of application variables are to be substituted when the statement is processed. The markers are referenced sequentially, from left to right, starting at 1.

When an application variable is associated with a parameter marker, it is *bound* to the parameter marker. Binding is carried out by calling the SQLSetParam() function with:

- The number of the parameter marker
- A pointer to the application variable
- The SQL type of the parameter
- The data type and length of the variable.

The application variable is called a *deferred* argument because only the pointer is passed when SQLSetParam() is called. No data is read from the variable until the statement is processed. This applies to both buffer arguments and arguments that indicate the length of the data in the buffer. Deferred arguments allow the application to modify the contents of the bound parameter variables, and repeat the processing of the statement with the new values.

When calling SQLSetParam(), it is possible to bind a variable of a different type from that required by the SQL statement. In this case DB2 UDB CLI converts the contents of the bound variable to the correct type. For example, the SQL statement might require an integer value, but your application has a string representation of an integer. The string can be bound to the parameter, and DB2 UDB CLI converts the string to an integer when you process the statement.

For more information and examples refer to:

- "Data types and data conversion in DB2 UDB CLI functions" on page 15
- "SQLPrepare - Prepare a statement" on page 164
- "SQLSetParam - Set parameter" on page 200
- "SQLExecute - Execute a statement" on page 83
- "SQLExecDirect - Execute a statement directly" on page 81

If the SQL statement uses parameter markers instead of expressions (or host variables in embedded SQL), you must bind the application variable to the parameter marker.

## Processing results in a DB2 UDB CLI application

The next step after the statement has been processed depends on the type of SQL statement.

**Process SELECT statements in a DB2 UDB CLI application:** If the statement is a SELECT, the following steps are generally needed to retrieve each row of the result set:

1. Establish the structure of the result set, number of columns, column types and lengths.
2. Bind application variables to columns in order to receive the data.
3. Repeatedly fetch the next row of data, and receive it into the bound application variables.

Columns that were not previously bound can be retrieved by calling SQLGetData() after each successful fetch.

**Note:** Each of the above steps requires some diagnostic checks.

The first step requires analyzing the processed or prepared statement. If the SQL statement is generated by the application, this step is not necessary. This is because the application knows the structure of the result set and the data types of each column. If the SQL statement is generated (for example, entered by a user) at run time, the application needs to query:

- The number of columns
- The type of each column
- The names of each column in the result set.

This information can be obtained by calling `SQLNumResultCols()` and `SQLDescribeCol()` (or `SQLColAttributes()`) after preparing the statement or after executing the statement.

The second step allows the application to retrieve column data directly into an application variable on the next call to `SQLFetch()`. For each column to be retrieved, the application calls `SQLBindCol()` to bind an application variable to a column in the result set. Similar to variables bound to parameter markers using `SQLSetParam()`, columns are bound using deferred arguments. This time the variables are output arguments, and data is written to them when `SQLFetch()` is called. `SQLGetData()` can also be used to retrieve data, so calling `SQLBindCol()` is optional.

The third step is to call `SQLFetch()` to fetch the first or next row of the result set. If any columns have been bound, the application variable is updated. If any data conversion is indicated by the data types specified on the call to `SQLBindCol`, the conversion occurs when `SQLFetch()` is called.

The last (optional) step, is to call `SQLGetData()` to retrieve any columns that were not previously bound. All columns can be retrieved this way, provided they were not bound, or a combination of both methods can be used. `SQLGetData()` is also useful for retrieving variable length columns in smaller pieces, which cannot be done with bound columns. Data conversion can also be indicated here, as in `SQLBindCol()`.

For more information and examples refer to:

- “Data types and data conversion in DB2 UDB CLI functions” on page 15
- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLColAttributes - Obtain column attributes” on page 52
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLFetch - Fetch next row” on page 87
- “SQLGetData - Get data from a column” on page 114
- “SQLNumResultCols - Get number of result columns” on page 160

**Process UPDATE, DELETE, and INSERT statements in a DB2 UDB CLI application:** If the statement is modifying data (UPDATE, DELETE or INSERT), no action is required other than the normal check for diagnostic messages. In this case, `SQLRowCount()` can be used to obtain the number of rows affected by the SQL statement.

If the SQL statement is a Positioned UPDATE or DELETE, it is necessary to use a *cursor*. A cursor is a moveable pointer to a row in the result table of a SELECT statement. In embedded SQL, cursors are used to retrieve, update or delete rows. When using DB2 UDB CLI, it is not necessary to define a cursor, because one is generated automatically.

In the case of Positioned UPDATE or DELETE statements, you need to specify the name of the cursor within the SQL statement. You can either define your own cursor name using `SQLSetCursorName()`, or query the name of the generated cursor using `SQLGetCursorName()`. It is best to use the generated name, because all error messages refer to this name, and not the one defined by `SQLSetCursorName()`.

#### Related reference

“SQLNumResultCols - Get number of result columns” on page 160

**Process other SQL statements in a DB2 UDB CLI application:** If the statement neither queries nor modifies the data, then there is no further action other than the normal check for diagnostic messages.

## Free statement handles in a DB2 UDB CLI application

Call `SQLFreeStmt()` to end processing for a particular statement handle. This function can be used to do one or more of the following tasks:

- Unbind all columns
- Unbind all parameters
- Close any cursors and discard the results
- Drop the statement handle, and release all associated resources

The statement handle can be reused provided it is not dropped.

## Commit or roll back in a DB2 UDB CLI application

The last step is to either commit or roll back the *transaction*, using `SQLTransact()`.

A transaction is a recoverable unit of work, or a group of SQL statements that can be treated as one atomic operation. This means that all the operations within the group are to be completed (committed) or undone (rolled back), as if they were a single operation.

When using DB2 UDB CLI, transactions are started implicitly with the first access to the database using `SQLPrepare()`, `SQLExecDirect()` or `SQLGetTypeInfo()`. The transaction ends when you use `SQLTransact()` to either roll back or commit the transaction. This means that any SQL statements processed between these are treated as one unit of work.

**When to call `SQLTransact()` in a DB2 UDB CLI application:** Consider the following items when deciding when to end a transaction:

- You can only commit or roll back the current transaction, so keep dependent statements within the same transaction.
- Various locks are held while you have an outstanding transaction. Ending the transaction releases the locks, and allows access to the data by other users. This is the case for all SQL statements, including `SELECT` statements.
- As soon as a transaction has successfully been committed or rolled back, it is fully recoverable from the system logs (this depends on the Database Management System (DBMS)). Open transactions are not recoverable.

**Effects of calling `SQLTransact()` in a DB2 UDB CLI application:** When a transaction ends:

- All statements must be prepared before they can be used again.
- Cursor names, bound parameters, and column bindings are maintained from one transaction to the next.
- All open cursors are closed.

### Related reference

“`SQLTransact` - Commit or roll back transaction” on page 217

## Diagnostics in a DB2 UDB CLI application

This topic deals with warning or error conditions generated within an application.

There are two levels of diagnostics when calling DB2 UDB CLI functions:

- Return codes from a DB2 UDB CLI application
- DB2 UDB CLI `SQLSTATE`s (diagnostic messages)

### Related reference

“`SQLError` - Retrieve error information” on page 79

“`SQLGetDiagField` - Return diagnostic information (extensible)” on page 119

## Return codes from a DB2 UDB CLI application

The following table lists all possible return codes for DB2 UDB CLI functions. Each function description in “DB2 UDB CLI functions” on page 18 lists the possible codes returned for each function.

Table 2. DB2 UDB CLI function return codes

Return code	Value	Explanation
SQL_SUCCESS	0	The function is completed successfully, no additional SQLSTATE information available.
SQL_SUCCESS_WITH_INFO	1	The function is completed successfully, with a warning or other information. Call <code>SQLError()</code> to receive the SQLSTATE and any other error information. The SQLSTATE has a class of 01.
SQL_NO_DATA_FOUND	100	The function returned successfully, but no relevant data is found.
SQL_ERROR	-1	The function fails. Call <code>SQLError()</code> to receive the SQLSTATE and any other error information.
SQL_INVALID_HANDLE	-2	The function fails because an input handle is not valid (environment, connection or statement handle).

## DB2 UDB CLI SQLSTATES

Because different database servers often have different diagnostic message codes, DB2 UDB CLI provides a standard set of *SQLSTATES* that are defined by the X/Open SQL CAE specification. This allows consistent message handling across different database servers.

SQLSTATES are alphanumeric strings of 5 characters (bytes) with a format of *ccsss*, where *cc* indicates class and *sss* indicates subclass. Any SQLSTATE that has a class of:

- 01, is a warning.
- HY, is generated by the CLI driver (either DB2 UDB CLI or ODBC).

The `SQLError()` function also returns an error code if the code is generated by the server. When connected to an IBM database server, the error code is `SQLCODE`. If the code is generated by DB2 UDB CLI instead of at the server, then the error code is set to `-99999`.

DB2 UDB CLI SQLSTATES include both additional IBM defined SQLSTATES that are returned by the database server, and DB2 UDB CLI defined SQLSTATES for conditions that are not defined in the X/Open specification. This allows for the maximum amount of diagnostic information to be returned. When running applications in Windows using ODBC, it is also possible to receive ODBC defined SQLSTATES.

Follow these guidelines for using SQLSTATES within your application:

- Always check the function return code before calling `SQLError()` to determine if diagnostic information is available.
- Use the SQLSTATES rather than the error code.
- To increase your application’s portability, only build dependencies on the subset of DB2 UDB CLI SQLSTATES that are defined by the X/Open specification, and return the additional ones as information only. (Dependencies refers to the application making logic flow decisions based on specific SQLSTATES.)
- For maximum diagnostic information, return the text message along with the SQLSTATE (if applicable, the text message includes the IBM defined SQLSTATE). It is also useful for the application to print out the name of the function that returned the error.

## Data types and data conversion in DB2 UDB CLI functions

This topics shows all of the supported SQL types and their corresponding symbolic names.

Table 3 shows all of the supported SQL types and their corresponding symbolic names. The symbolic names are used in `SQLBindParam()`, `SQLBindParameter()`, `SQLSetParam()`, `SQLBindCol()`, and `SQLGetData()` to indicate the data types of the arguments.

Each column is described as follows:

**SQL type**

This column contains the SQL data type as it appears in an SQL statement. The SQL data types are dependent on the Database Management System (DBMS).

**SQL symbolic**

This column contains an SQL symbolic name that is defined (in `sqlcli.h`) as an integer value. This value is used by various functions to identify an SQL data type in the first column.

*Table 3. SQL data types and default C data types*

SQL type	SQL symbolic
BIGINT	SQL_BIGINT
BINARY	SQL_BINARY
BLOB	SQL_BLOB
CHAR	SQL_CHAR, SQL_WCHAR <sup>1</sup>
CLOB	SQL_CLOB
DATE	SQL_DATE
DBCLOB	SQL_DBCLOB
DECIMAL	SQL_DECIMAL
DOUBLE	SQL_DOUBLE
FLOAT	SQL_FLOAT
GRAPHIC	SQL_GRAPHIC
INTEGER	SQL_INTEGER
NUMERIC	SQL_NUMERIC
REAL	SQL_REAL
SMALLINT	SQL_SMALLINT
TIME	SQL_TIME
TIMESTAMP	SQL_TIMESTAMP
VARBINARY	SQL_VARBINARY
VARCHAR	SQL_VARCHAR, SQL_WVARCHAR <sup>1</sup>
VARGRAPHIC	SQL_VARGRAPHIC
<sup>1</sup> SQL_WCHAR and SQL_WVARCHAR can be used to indicate Unicode data.	

**Other C data types in DB2 UDB CLI functions**

As well as the data types that map to SQL data types, there are also C symbolic types used for other function arguments, such as pointers and handles.

*Table 4. Generic data types and actual C data types*

Symbolic type	Actual C type	Typical usage
SQLHDBC	long int	Handle referencing database connection information.
SQLHENV	long int	Handle referencing environment information.
SQLHSTMT	long int	Handle referencing statement information.



Table 4. Generic data types and actual C data types (continued)

Symbolic type	Actual C type	Typical usage
SQLPOINTER	void *	Pointers to storage for data and parameters.
SQLRETURN	long int	Return code from DB2 UDB CLI functions.

## Data conversion in DB2 UDB CLI functions

As mentioned previously, DB2 UDB CLI manages the transfer and any required conversion of data between the application and the Database Management System (DBMS). Before the data transfer actually takes place, the source, target or both data types are indicated when calling `SQLBindParam()`, `SQLBindParameter()`, `SQLSetParam()`, `SQLBindCol()` or `SQLGetData()`. These functions use the symbolic type names shown in Table 3 on page 16, to identify the data types involved. Refer to the “SQLFetch - Fetch next row” on page 87, or “SQLGetCol - Retrieve one column of a row of the result set” on page 103 for examples of the functions that use the symbolic data types.

For a list of supported data type conversions in DB2 UDB CLI, see the data type compatibility table in the Assignments and comparisons topic. Other conversions can be achieved by using SQL scalar functions or the SQL CAST function in the SQL syntax of the statement being processed.

The functions mentioned in the previous paragraph can be used to convert data to other types. Not all data conversions are supported or make sense.

Whenever truncation that is rounding or data type incompatibilities occur on a function call, either `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO` is returned. Further information is then indicated by the `SQLSTATE` value and other information returned by `SQLError()`.

## Work with string arguments in DB2 UDB CLI functions

These topics discuss some conventions when dealing with the various aspects of working with string arguments in DB2 UDB CLI functions.

### Length of string arguments in DB2 UDB CLI functions

Input string arguments have an associated length argument. This argument indicates to DB2 UDB CLI, either the length of the allocated buffer (not including the null byte terminator), or the special value `SQL_NTS`. If `SQL_NTS` is passed, DB2 UDB CLI determines the length of the string by locating the null terminating character.

Output string arguments have two associated length arguments, one to specify the length of the allocated buffer and one to return the length of the string returned by DB2 UDB CLI. The returned length value is the total length of the string available for return, whether it fits in the buffer or not.

For SQL column data, if the output is an empty string, `SQL_NULL_DATA` is returned in the length argument.

If a function is called with a null pointer for an output length argument, DB2 UDB CLI does not return a length. This might be useful when it is known that the buffers are large enough for all possible results. If DB2 UDB CLI attempts to return the `SQL_NULL_DATA` value to indicate a column contains null data and the output length argument is a null pointer, the function call fails.

Every character string that DB2 UDB CLI returns is terminated with a null terminating character (hexadecimal 00), except for strings that are returned from graphic data types. This requires that all buffers allocate enough space for the maximum number that are expected, plus one for the null-terminating character.

## String truncation in DB2 UDB CLI functions

If an output string does not fit into a buffer, DB2 UDB CLI truncates the string to a length that is one less than the size of the buffer, and writes the null terminator. If truncation occurs, the function returns `SQL_SUCCESS_WITH_INFO` and an `SQLSTATE` by indicating truncation. The application can then compare the buffer length to the output length to determine which string is truncated.

For example, if `SQLFetch()` returns `SQL_SUCCESS_WITH_INFO`, and an `SQLSTATE` of 01004, at least one of the buffers bound to a column is too small to hold the data. For each buffer that is bound to a column, the application can compare the buffer length with the output length and determine which column is truncated.

## Interpretation of strings in DB2 UDB CLI functions

DB2 UDB CLI ignores case, and removes leading and trailing blanks for all string input arguments, such as column names and cursor names, with the exception of:

- Any database data
- Delimited identifiers that are enclosed in double quotation marks)
- Password arguments.

---

## DB2 UDB CLI functions

This topic provides a description of each CLI function.

See “Categories of DB2 UDB CLIs” on page 19 for a categorical listing of the functions.

Each of the DB2 UDB CLI function descriptions is presented in a consistent format.

### How the CLI functions are described

The following table shows the type of information that is described in each section of the function description.

Type	Description
Purpose	This section gives a brief overview of what the function does. It also indicates if any functions should be called before and after calling the function being described.
Syntax	This section contains the C language prototype for the i5/OS™ environment.
Arguments	<p>This section lists each function argument, along with its data type, a description and whether it is an input or output argument.</p> <p>Each DB2 UDB CLI argument is either an input or output argument. With the exception of <code>SQLGetInfo()</code>, DB2 UDB CLI only modifies arguments that are indicated as output.</p> <p>Some functions contain input or output arguments which are known as <i>deferred</i> or <i>bound</i> arguments. These arguments are pointers to buffers allocated by the application. These arguments are associated with (or bound to) either a parameter in an SQL statement, or a column in a result set. The data areas specified by the function are accessed by DB2 UDB CLI at a later time. It is important that these deferred data areas are still valid at the time DB2 UDB CLI accesses them.</p>
Usage	This section provides information about how to use the function, and any special considerations. Possible error conditions are not discussed here, but are listed in the diagnostics section instead.



Type	Description
Return codes	<p>This section lists all the possible function return codes. When SQL_ERROR or SQL_SUCCESS_WITH_INFO is returned, error information can be obtained by calling SQLError().</p> <p>Refer to “Diagnostics in a DB2 UDB CLI application” on page 14 for more information about return codes.</p>
Diagnostics	<p>This section contains a table that lists the SQLSTATEs explicitly returned by DB2 UDB CLI (SQLSTATEs generated by the Database Management System (DBMS) might also be returned) and indicates the cause of the error. These values are obtained by calling SQLError() after the function returns SQL_ERROR or SQL_SUCCESS_WITH_INFO.</p> <p>An * in the first column indicates that the SQLSTATE is returned only by DB2 UDB CLI, and is not returned by other ODBC drivers.</p> <p>Refer to “Diagnostics in a DB2 UDB CLI application” on page 14 for more information about diagnostics.</p>
Restrictions	<p>This section indicates any differences or limitations between DB2 UDB CLI and ODBC that might affect an application.</p>
Example	<p>This section is a code fragment demonstrating the use of the function. The complete source used for all code fragments is listed in “Examples: DB2 UDB CLI applications” on page 247.</p>
References	<p>This section lists related DB2 UDB CLI functions.</p>

## Categories of DB2 UDB CLIs

The following call level interface APIs are available for database access on iSeries. Each of the DB2 UDB CLI function descriptions is presented in a consistent format.

- **Connect**
  - “SQLConnect - Connect to a data source” on page 62
  - “SQLDataSources - Get list of data sources” on page 65
  - “SQLDisconnect - Disconnect from a data source” on page 73
  - “SQLDriverConnect - (Expanded) Connect to a data source” on page 74
- **Diagnostics**
  - “SQLError - Retrieve error information” on page 79
  - “SQLGetDiagField - Return diagnostic information (extensible)” on page 119
  - “SQLGetDiagRec - Return diagnostic information (concise)” on page 121
- **MetaData**
  - “SQLColumns - Get column information for a table” on page 58
  - “SQLColumnPrivileges - Get privileges associated with the columns of a table” on page 56
  - “SQLForeignKeys - Get the list of foreign key columns” on page 94
  - “SQLGetInfo - Get general information” on page 127
  - “SQLGetTypeInfo - Get data type information” on page 149
  - “SQLLanguages - Get SQL dialect or conformance information” on page 153
  - “SQLPrimaryKeys - Get primary key columns of a table” on page 168
  - “SQLProcedureColumns - Get input/output parameter information for a procedure” on page 170
  - “SQLProcedures - Get list of procedure names” on page 176
  - “SQLSpecialColumns - Get special (row identifier) columns” on page 206
  - “SQLStatistics - Get index and statistics information for a base table” on page 209
  - “SQLTablePrivileges - Get privileges associated with a table” on page 212

- “SQLTables - Get table information” on page 215
- **Process SQL statements**
  - “SQLBindCol - Bind a column to an application variable” on page 29
  - “SQLBindFileToCol - Bind LOB file reference to LOB column” on page 33
  - “SQLBindFileToParam - Bind LOB file reference to LOB parameter” on page 36
  - “SQLBindParam - Bind a buffer to a parameter marker” on page 38
  - “SQLBindParameter - Bind a parameter marker to a buffer” on page 42
  - “SQLCancel - Cancel statement” on page 50
  - “SQLCloseCursor - Close cursor statement” on page 51
  - “SQLColAttributes - Obtain column attributes” on page 52
  - “SQLDescribeCol - Describe column attributes” on page 68
  - “SQLDescribeParam - Return description of a parameter marker” on page 70
  - “SQLEndTran - Commit or roll back a transaction” on page 77
  - “SQLExecDirect - Execute a statement directly” on page 81
  - “SQLExecute - Execute a statement” on page 83
  - “SQLExtendedFetch - Fetch array of rows” on page 85
  - “SQLFetch - Fetch next row” on page 87
  - “SQLFetchScroll - Fetch from a scrollable cursor” on page 92
  - “SQLGetCursorName - Get cursor name” on page 111
  - “SQLGetData - Get data from a column” on page 114
  - “SQLGetDescField - Get descriptor field” on page 115
  - “SQLGetDescRec - Get descriptor record” on page 117
  - “SQLMoreResults - Determine whether there are more result sets” on page 155
  - “SQLNativeSql - Get native SQL text” on page 156
  - “SQLNextResult - Process the next result set” on page 158
  - “SQLNumParams - Get number of parameters in an SQL statement” on page 159
  - “SQLNumResultCols - Get number of result columns” on page 160
  - “SQLParamData - Get next parameter for which a data value is needed” on page 162
  - “SQLParamOptions - Specify an input array for a parameter” on page 163
  - “SQLPrepare - Prepare a statement” on page 164
  - “SQLPutData - Pass data value for a parameter” on page 179
  - “SQLRowCount - Get row count” on page 182
  - “SQLSetCursorName - Set cursor name” on page 192
  - “SQLTransact - Commit or roll back transaction” on page 217
- **Work with attributes**
  - “SQLGetCol - Retrieve one column of a row of the result set” on page 103
  - “SQLGetConnectAttr - Get the value of a connection attribute” on page 109
  - “SQLGetConnectOption - Return current setting of a connect Option” on page 110
  - “SQLGetCursorName - Get cursor name” on page 111
  - “SQLGetData - Get data from a column” on page 114
  - “SQLGetDescField - Get descriptor field” on page 115
  - “SQLGetDescRec - Get descriptor record” on page 117
  - “SQLGetEnvAttr - Return current setting of an environment Attribute” on page 123
  - “SQLGetFunctions - Get functions” on page 124
  - “SQLGetInfo - Get general information” on page 127

- "SQLGetLength - Retrieve length of a string value" on page 140
- "SQLGetPosition - Return starting position of string" on page 141
- "SQLGetStmtAttr - Get the value of a statement attribute" on page 143
- "SQLGetStmtOption - Return current setting of a statement option" on page 145
- "SQLGetSubString - Retrieve portion of a string value" on page 146
- "SQLGetTypeInfo - Get data type information" on page 149
- "SQLSetConnectAttr - Set a connection attribute" on page 183
- "SQLSetConnectOption - Set connection option" on page 190
- "SQLSetCursorName - Set cursor name" on page 192
- "SQLSetDescField - Set a descriptor field" on page 193
- "SQLSetDescRec - Set a descriptor record" on page 195
- "SQLSetEnvAttr - Set environment attribute" on page 196
- "SQLSetParam - Set parameter" on page 200
- "SQLSetStmtAttr - Set a statement attribute" on page 201
- "SQLSetStmtOption - Set statement option" on page 205
- **Work with handles**
  - "SQLAllocConnect - Allocate connection handle"
  - "SQLAllocEnv - Allocate environment handle" on page 24
  - "SQLAllocHandle - Allocate handle" on page 26
  - "SQLAllocStmt - Allocate a statement handle" on page 27
  - "SQLCopyDesc - Copy description statement" on page 64
  - "SQLFreeConnect - Free connection handle" on page 99
  - "SQLFreeEnv - Free environment handle" on page 100
  - "SQLFreeHandle - Free a handle" on page 101
  - "SQLFreeStmt - Free (or reset) a statement handle" on page 102
  - "SQLReleaseEnv - Release all environment resources" on page 181

## SQLAllocConnect - Allocate connection handle

### Purpose

SQLAllocConnect() allocates a connection handle and associated resources within the environment identified by the input environment handle. Call SQLGetInfo() with fInfoType set to SQL\_ACTIVE\_CONNECTIONS, to query the number of connections that can be allocated at any one time.

SQLAllocEnv() must be called before calling this function.

### Syntax

```
SQLRETURN SQLAllocConnect (SQLHENV    henv,
                           SQLHDBC    *phdbc);
```

### Function arguments

Table 5. SQLAllocConnect arguments

Data type	Argument	Use	Description
SQLHDBC *	<i>phdbc</i>	Output	Pointer to connection handle
SQLHENV	<i>henv</i>	Input	Environment handle

## Usage

The output connection handle is used by DB2 UDB CLI to reference all information related to the connection, including general status information, transaction state, and error information.

If the pointer to the connection handle (*phdbc*) points to a valid connection handle allocated by `SQLAllocConnect()`, the original value is overwritten as a result of this call. This is an application programming error and is not detected by DB2 UDB CLI

## Return codes

- `SQL_SUCCESS`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

If `SQL_ERROR` is returned, the *phdbc* argument is set to `SQL_NULL_HDBC`. The application should call `SQLError()` with the environment handle (*henv*), with *hdbc* set to `SQL_NULL_HDBC`, and with *hstmt* set to `SQL_NULL_HSTMT`.

## Diagnostics

Table 6. `SQLAllocConnect` `SQLSTATE`s

CLI <code>SQLSTATE</code>	Description	Explanation
<code>HY001</code>	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
<code>HY009</code>	Argument value that is not valid	<i>phdbc</i> is a null pointer.

## Example

The following example shows how to obtain diagnostic information for the connection and the environment. For more examples of using `SQLError()`, refer to “Example: Interactive SQL and the equivalent DB2 UDB CLI function calls” on page 253 for a complete listing of typical.c.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```
/******  
** initialize  
** - allocate environment handle  
** - allocate connection handle  
** - prompt for server, user id, & password  
** - connect to server  
*****/  
  
int initialize(SQLHENV *henv,  
              SQLHDBC *hdbc)  
{  
    SQLCHAR    server[SQL_MAX_DSN_LENGTH],  
              uid[30],  
              pwd[30];  
    SQLRETURN  rc;  
  
    SQLAllocEnv (henv);          /* allocate an environment handle */  
    if (rc != SQL_SUCCESS )  
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
```

```

SQLAllocConnect (*henv, hdbc); /* allocate a connection handle */
if (rc != SQL_SUCCESS )
    check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

printf("Enter Server Name:\n");
gets(server);
printf("Enter User Name:\n");
gets(uid);
printf("Enter Password Name:\n");
gets(pwd);

if (uid[0] == '\0')
{
    rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
}
else
{
    rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
}
}/* end initialize */

/*****
int check_error (SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt,
                SQLRETURN   frc)
{
SQLRETURN   rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
case SQL_SUCCESS : break;
case SQL_ERROR :
case SQL_INVALID_HANDLE:
    printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
    rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
    if (rc != SQL_SUCCESS)
        printf("Rollback Failed, Exiting application\n");
    else
        printf("Rollback Successful, Exiting application\n");
    terminate(henv, hdbc);
    exit(frc);
    break;
case SQL_SUCCESS_WITH_INFO :
    printf("\n ** Warning Message, application continuing\n");
    break;
case SQL_NO_DATA_FOUND :
    printf("\n ** No Data Found ** \n");
    break;
default :
    printf("\n ** Invalid Return Code ** \n");
    printf(" ** Attempting to rollback transaction **\n");
    SQLTransact(henv, hdbc, SQL_ROLLBACK);
    terminate(henv, hdbc);
    exit(frc);
    break;
}
return(SQL_SUCCESS);
}

```

## References

- “SQLAllocEnv - Allocate environment handle”
- “SQLConnect - Connect to a data source” on page 62
- “SQLDisconnect - Disconnect from a data source” on page 73
- “SQLFreeConnect - Free connection handle” on page 99
- “SQLGetConnectAttr - Get the value of a connection attribute” on page 109
- “SQLSetConnectOption - Set connection option” on page 190

## SQLAllocEnv - Allocate environment handle

### Purpose

SQLAllocEnv() allocates an environment handle and associated resources.

An application must call this function before SQLAllocConnect() or any other DB2 UDB CLI functions. The *henv* value is passed in all later function calls that require an environment handle as input.

### Syntax

```
SQLRETURN SQLAllocEnv (SQLHENV *phenv);
```

### Function arguments

Table 7. SQLAllocEnv arguments

Data type	Argument	Use	Description
SQLHENV *	<i>phenv</i>	Output	Pointer to environment handle

### Usage

There can be only one active environment at any one time per application. Any later call to SQLAllocEnv() returns the existing environment handle.

By default, the first successful call to SQLFreeEnv() releases the resources associated with the handle. This occurs no matter how many times SQLAllocEnv() is successfully called. If the environment attribute SQL\_ATTR\_ENVHNDL\_COUNTER is set to SQL\_TRUE, SQLFreeEnv() must be called once for each successful SQLAllocEnv() call before the resources associated with the handle are released.

To ensure that all DB2 UDB CLI resources are kept active, the program that calls SQLAllocEnv() should not stop or leave the stack. Otherwise, the application loses open cursors, statement handles, and other resources it has allocated.

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR

If SQL\_ERROR is returned and *phenv* is equal to SQL\_NULL\_HENV, then SQLError() cannot be called because there is no handle with which to associate additional diagnostic information.

If the return code is SQL\_ERROR and the pointer to the environment handle is not equal to SQL\_NULL\_HENV, then the handle is a *restricted handle*. This means the handle can only be used in a call to SQLError() to obtain more error information, or to SQLFreeEnv().

## Diagnostics

Table 8. *SQLAllocEnv SQLSTATES*

SQLSTATE	Description	Explanation
58004	System error	Unrecoverable system error

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```
/******  
** file = basiccon.c  
** - demonstrate basic connection to two datasources.  
** - error handling ignored for simplicity  
**  
** Functions used:  
**  
**   SQLAllocConnect  SQLDisconnect  
**   SQLAllocEnv      SQLFreeConnect  
**   SQLConnect       SQLFreeEnv  
**  
**  
*****/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include "sqlcli.h"  
  
int  
connect(SQLHENV henv,  
        SQLHDBC * hdbc);  
  
#define MAX_DSN_LENGTH  18  
#define MAX_UID_LENGTH  10  
#define MAX_PWD_LENGTH  10  
#define MAX_CONNECTIONS 5  
  
int  
main()  
{  
    SQLHENV      henv;  
    SQLHDBC      hdbc[MAX_CONNECTIONS];  
  
    /* allocate an environment handle */  
    SQLAllocEnv(&henv);  
  
    /* Connect to first data source */  
    connect(henv, &hdbc[0]);  
  
    /* Connect to second data source */  
    connect(henv, &hdbc[1]);  
  
    /****** Start Processing Step *****/  
    /* allocate statement handle, execute statement, and so on */  
    /****** End Processing Step *****/  
  
    printf("\nDisconnecting ....\n");  
    SQLFreeConnect(hdbc[0]); /* free first connection handle */  
    SQLFreeConnect(hdbc[1]); /* free second connection handle */  
    SQLFreeEnv(henv); /* free environment handle */  
  
    return (SQL_SUCCESS);  
}
```

```

/*****
** connect - Prompt for connect options and connect      **
*****/

int
connect(SQLHENV henv,
        SQLHDBC * hdbc)
{
    SQLRETURN      rc;
    SQLCHAR        server[MAX_DSN_LENGTH + 1], uid[MAX_UID_LENGTH + 1],
pwd[MAX_PWD_LENGTH
+ 1];
    SQLCHAR        buffer[255];
    SQLSMALLINT    outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    SQLAllocConnect(henv, hdbc); /* allocate a connection handle */

    rc = SQLConnect(*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database\n");
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
        return (SQL_SUCCESS);
    }
}

```

## References

- “SQLAllocConnect - Allocate connection handle” on page 21
- “SQLFreeEnv - Free environment handle” on page 100
- “SQLAllocStmt - Allocate a statement handle” on page 27

## SQLAllocHandle - Allocate handle

### Purpose

SQLAllocHandle() allocates any type of handle.

### Syntax

```

SQLRETURN SQLAllocHandle (SQLSMALLINT htype,
                        SQLINTEGER ihandle,
                        SQLINTEGER *handle);

```

### Function arguments

Table 9. SQLAllocHandle arguments

Data type	Argument	Use	Description
SQLINTEGER *	<i>handle</i>	Output	Pointer to the handle.
SQLINTEGER	<i>ihandle</i>	Input	The handle that describes the context in which the new handle is allocated; however, if <i>htype</i> is SQL_HANDLE_ENV, this is SQL_NULL_HANDLE.



Table 9. *SQLAllocHandle* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>htype</i>	Input	Type of handle to allocate. Must be either SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_DESC, or SQL_HANDLE_STMT.

## Usage

This function combines the functions of `SQLAllocEnv()`, `SQLAllocConnect()`, and `SQLAllocStmt()`.

If *htype* is `SQL_HANDLE_ENV`, *ihandle* must be `SQL_NULL_HANDLE`. If *htype* is `SQL_HANDLE_DBC`, *ihandle* must be a valid environment handle. If *htype* is either `SQL_HANDLE_DESC` or `SQL_HANDLE_STMT`, *ihandle* must be a valid connection handle.

## Return codes

- `SQL_SUCCESS`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

## Diagnostics

`SQL_ERROR` is returned if the argument handle is a null pointer.

Table 10. *SQLAllocHandle* SQLSTATEs

SQLSTATE	Description	Explanation
58004	System error	Unrecoverable system error.
HY014	Too many handles	The maximum number of handles has been allocated.

## References

- “`SQLAllocConnect` - Allocate connection handle” on page 21
- “`SQLAllocEnv` - Allocate environment handle” on page 24
- “`SQLAllocStmt` - Allocate a statement handle”

## SQLAllocStmt - Allocate a statement handle

### Purpose

`SQLAllocStmt()` allocates a new statement handle and associates it with the connection specified by the connection handle. There is no defined limit on the number of statement handles that can be allocated at any one time.

`SQLConnect()` must be called before calling this function.

This function must be called before `SQLBindParam()`, `SQLPrepare()`, `SQLExecute()`, `SQLExecDirect()`, or any other function that has a statement handle as one of its input arguments.

### Syntax

```
SQLRETURN SQLAllocStmt (SQLHDBC hdbc,
                        SQLHSTMT *phstmt);
```

## Function arguments

Table 11. *SQLAllocStmt* arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Connection handle
SQLHSTMT *	<i>phstmt</i>	Output	Pointer to statement handle

## Usage

DB2 UDB CLI uses each statement handle to relate all the descriptors, result values, cursor information, and status information to the SQL statement processed. Although each SQL statement must have a statement handle, you can reuse the handles for different statements.

A call to this function requires that *hdbc* references an active database connection.

To process a positioned update or delete, the application must use different statement handles for the SELECT statement and the UPDATE or DELETE statement.

If the input pointer to the statement handle (*phstmt*) points to a valid statement handle allocated by a previous call to *SQLAllocStmt()*, then the original value is overwritten as a result of this call. This is an application programming error and is not detected by DB2 UDB CLI.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

If SQL\_ERROR is returned, the *phstmt* argument is set to SQL\_NULL\_HSTMT. The application should call *SQLError()* with the same *hdbc* and with the *hstmt* argument set to SQL\_NULL\_HSTMT.

## Diagnostics

Table 12. *SQLAllocStmt* SQLSTATEs

SQLSTATE	Description	Explanation
08003	Connection not open	The connection specified by the <i>hdbc</i> argument is not open. The connection must be established successfully (and the connection must be open) for the driver to allocate an <i>hstmt</i> .
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	<i>phstmt</i> is a null pointer.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## Example

Refer to the example in “SQLFetch - Fetch next row” on page 87.

## References

- “SQLConnect - Connect to a data source” on page 62
- “SQLFreeStmt - Free (or reset) a statement handle” on page 102
- “SQLGetStmtOption - Return current setting of a statement option” on page 145
- “SQLSetStmtOption - Set statement option” on page 205

## SQLBindCol - Bind a column to an application variable

### Purpose

SQLBindCol() associates (bind) columns in a result set to application variables (storage buffers), for all data types. Data is transferred from the Database Management System (DBMS) to the application when SQLFetch() is called.

This function is also used to specify any data conversion required. It is called once for each column in the result set that the application needs to retrieve.

SQLPrepare() or SQLExecDirect() is typically called before this function. It might also be necessary to call SQLDescribeCol() or SQLColAttributes().

SQLBindCol() must be called before SQLFetch(), to transfer data to the storage buffers specified by this call.

### Syntax

```
SQLRETURN SQLBindCol (SQLHSTMT      hstmt,  
                     SQLSMALLINT    icol,  
                     SQLSMALLINT    fCType,  
                     SQLPOINTER     rgbValue,  
                     SQLINTEGER     cbValueMax,  
                     SQLINTEGER     *pcbValue);
```

### Function arguments

Table 13. SQLBindCol arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER *	<i>pcbValue</i>	Output (deferred)	Pointer to value which indicates the number of bytes DB2 UDB CLI has available to return in the <i>rgbValue</i> buffer.  SQLFetch() returns SQL_NULL_DATA in this argument if the data value of the column is null. SQL_NTS is returned in this argument if the data value of the column is returned as a null-terminated string.

Table 13. SQLBindCol arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>cbValueMax</i>	Input	<p>Size of <i>rgbValue</i> buffer in bytes available to store the column data.</p> <p>If <i>fcType</i> is either SQL_CHAR or SQL_DEFAULT, then <i>cbValueMax</i> must be &gt; 0 otherwise an error is returned.</p> <p>If <i>fcType</i> is either SQL_DECIMAL or SQL_NUMERIC, <i>cbValueMax</i> must actually be a precision and scale. The method to specify both values is to use (<i>precision</i> * 256) + <i>scale</i>. This is also the value returned as the LENGTH of these data types when using SQLColAttributes().</p> <p>If <i>fcType</i> specifies any form of double-byte character data, then <i>cbValueMax</i> must be the number of double-byte characters, not the number of bytes.</p>
SQLPOINTER	<i>rgbValue</i>	Output (deferred)	<p>Pointer to buffer where DB2 UDB CLI is to store the column data when the fetch occurs.</p> <p>If <i>rgbValue</i> is null, the column is unbound.</p>

Table 13. *SQLBindCol* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>fCType</i>	Input	<p>Application data type for column number <i>icol</i> in the result set. The following types are supported:</p> <ul style="list-style-type: none"> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BLOB</li> <li>• SQL_BLOB_LOCATOR</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_CLOB_LOCATOR</li> <li>• SQL_DATALINK</li> <li>• SQL_DATETIME</li> <li>• SQL_DBCLOB</li> <li>• SQL_DBCLOB_LOCATOR</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TYPE_DATE</li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_VARBINARY</li> <li>• SQL_VARCHAR</li> <li>• SQL_VARGRAPHIC</li> <li>• SQL_WCHAR</li> <li>• SQL_WVARCHAR</li> </ul> <p>Specifying SQL_DEFAULT causes data to be transferred to its default data type; refer to Table 3 on page 16 for more information.</p>
SQLSMALLINT	<i>icol</i>	Input	<p>Number identifying the column. Columns are numbered sequentially, from left to right, starting at 1.</p>

**Note:**

For this function, both *rgbValue* and *pcbValue* are deferred outputs, meaning that the storage locations these pointers point to are not updated until `SQLFetch()` is called. The locations referred to by these pointers must remain valid until `SQLFetch()` is called.

## Usage

The application calls `SQLBindCol()` once for each column in the result set that it wants to retrieve. When `SQLFetch()` is called, the data in each of these *bound* columns is placed in the assigned location (given by the pointers *rgbValue* and *pcbValue*).

The application can query the attributes (such as data type and length) of the column by first calling `SQLDescribeCol()` or `SQLColAttributes()`. This information can then be used to specify the correct data type of the storage locations, or to indicate data conversion to other data types. Refer to “Data types and data conversion in DB2 UDB CLI functions” on page 15 for more information.

In later fetches, the application can change the binding of these columns or bind unbound columns by calling `SQLBindCol()`. The new binding does not apply to data fetched, it is used when the next `SQLFetch()` is called. To unbind a single column, call `SQLBindCol()` with *rgbValue* set to `NULL`. To unbind all the columns, the application should call `SQLFreeStmt()` with the *fOption* input set to `SQL_UNBIND`.

Columns are identified by a number, assigned sequentially from left to right, starting at 1. The number of columns in the result set can be determined by calling `SQLNumResultCols()` or `SQLColAttributes()` with the *fdescType* argument set to `SQL_DESC_COUNT`.

All character data is treated as the default job coded character set identifier (CCSID) if the `SQL_ATTR_UTF8` environment attribute is not set to `SQL_TRUE`.

An application can choose not to bind every column, or even not to bind any columns. The data in the unbound columns (and only the unbound columns) can be retrieved using `SQLGetData()` after `SQLFetch()` has been called. `SQLBindCol()` is more efficient than `SQLGetData()`, and should be used whenever possible.

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 requires; otherwise, the data might be truncated.

The default is null termination for output character strings. To change this you must set the `SQLSetEnvAttr()` attribute `SQL_ATTR_OUTPUT_NTS` to `SQL_FALSE`. The output values for *pcbValue* after a call to `SQLFetch()` behave in the following way for character data types:

- If the null termination attribute is set (the default) and no truncation occurs, then `SQL_NTS` is returned in the *pcbValue*.
- If the null termination attribute is not set and no truncation occurs, then the value of *cbValueMax* is returned in *pcbValue*.
- If the null termination attribute is set or not set and truncation occurs, then the value of *cbValueMax* is returned in *pcbValue*.

If truncation occurs and the `SQLSetEnvAttr()` attribute `SQL_ATTR_TRUNCATION_RTNC` is set to `SQL_FALSE` (which is the default), then `SQL_SUCCESS` is returned in the `SQLFetch()` return code. If truncation occurs and the attribute is `SQL_TRUE`, then `SQL_SUCCESS_WITH_INFO` is returned. `SQL_SUCCESS` is returned in both cases if no truncation occurs.

Truncation occurs when argument *cbValueMax* does not allocate space for the amount of fetched data. If the environment is set to run with null terminated strings, make sure to allocate space for the additional byte in *cbValueMax*. For additional truncation information, refer to “SQLFetch - Fetch next row” on page 87.

## Return codes

- `SQL_SUCCESS`
- `SQL_ERROR`

- SQL\_INVALID\_HANDLE

## Diagnostics

Table 14. SQLBindCol SQLSTATES

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY002	Column number that is not valid	The value specified for the argument <i>icol</i> is 0.  The value specified for the argument <i>icol</i> exceeded the maximum number of columns supported by the data source.
HY003	Program type out of range	<i>fCType</i> is not a valid data type.
HY009	Argument value that is not valid	<i>rgbValue</i> is a null pointer.  The value specified for the argument <i>cbValueMax</i> is less than 1, and the argument <i>fCType</i> is either SQL_CHAR or SQL_DEFAULT.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY014	Too many handles	The maximum number of handles has been allocated, and use of this function requires an additional descriptor handle.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYC00	Driver not capable	The driver recognizes, but does not support the data type specified in the argument <i>fCType</i> (see also HY003).

## Example

Refer to the example in “SQLFetch - Fetch next row” on page 87.

## References

- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLFetch - Fetch next row” on page 87
- “SQLPrepare - Prepare a statement” on page 164

## SQLBindFileToCol - Bind LOB file reference to LOB column

### Purpose

SQLBindFileToCol() is used to associate (bind) an LOB 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 fetch.

### Syntax

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

## Function arguments

Table 15. SQLBindFileToCol arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>FileName</i>	Input (deferred)	Pointer to the location that contains the file name or an array of file names at the time of the next fetch using the <i>StatementHandle</i> . 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.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLINTEGER *	<i>FileOptions</i>	Input (deferred)	Pointer to the location that contains the file option to be used when writing the file at the time of the next fetch using the <i>StatementHandle</i> . The following <i>FileOptions</i> are supported:  <b>SQL_FILE_CREATE</b> Create a new file. If a file by this name already exists, SQL_ERROR is returned.  <b>SQL_FILE_OVERWRITE</b> If the file already exists, overwrite it. Otherwise, create a new file.  <b>SQL_FILE_APPEND</b> 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.
SQLINTEGER *	<i>IndicatorValue</i>	Output (deferred)	Pointer to the location that contains an indicator value.
SQLINTEGER *	<i>StringLength</i>	Output (deferred)	Pointer to the location that contains the length in bytes of the LOB data that is returned. If this pointer is NULL, nothing is returned.
SQLSMALLINT *	<i>FileNameLength</i>	Input (deferred)	Pointer to the location that contains the length of the file name (or an array of lengths) at the time the next fetch using the <i>StatementHandle</i> . If this pointer is NULL, then a length of SQL_NTS is assumed.  The maximum value of the file name length is 255.
SQLSMALLINT	<i>ColumnNumber</i>	Input	Number identifying the column. Columns are numbered sequentially, from left to right, starting at 1.
SQLSMALLINT	<i>MaxFileNameLength</i>	Input	This specifies the length of the <i>FileName</i> buffer.



## 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.

*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 an 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.

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

## Error conditions

Table 16. `SQLBindFileToCol` `SQLSTATEs`

SQLSTATE	Description	Explanation
58004	Unexpected system failure	Unrecoverable system error.
HY002	Column number that is not valid	The value specified for the argument <i>icol</i> is less than 1.  The value specified for the argument <i>icol</i> exceeded the maximum number of columns supported by the data source.
HY009	Argument value that is not valid	<i>FileName</i> , <i>StringLength</i> , or <i>FileOptions</i> is a null pointer.
HY010	Function sequence error	The function is called while in a data-at-processing ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.  The function is called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value specified for the argument <i>MaxFileNameLength</i> is 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.

## References

- “`SQLBindCol` - Bind a column to an application variable” on page 29
- “`SQLFetch` - Fetch next row” on page 87
- “`SQLBindFileToParam` - Bind LOB file reference to LOB parameter” on page 36

## SQLBindFileToParam - Bind LOB file reference to LOB parameter

### Purpose

SQLBindFileToParam() is used to associate (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 an LOB column when that statement is subsequently processed.

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,
                              SQLSMALLINT       ParameterNumber,
                              SQLSMALLINT       DataType,
                              SQLCHAR           *FileName,
                              SQLSMALLINT       *FileNameLength,
                              SQLINTEGER        *FileOptions,
                              SQLSMALLINT       MaxFileNameLength,
                              SQLINTEGER        *IndicatorValue);
```

### Function arguments

Table 17. SQLBindFileToParam arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>FileName</i>	Input (deferred)	Pointer to the location that contains the file name or an array of file names when the statement ( <i>StatementHandle</i> ) is processed. 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.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLINTEGER *	<i>FileOptions</i>	Input (deferred)	Pointer to the location that contains the file option (or an array of file options) to be used when reading the file. The location is accessed when the statement ( <i>StatementHandle</i> ) is processed. Only one option is supported (and it must be specified):  <b>SQL_FILE_READ</b> A regular file that can be opened, read and closed. (The length is computed when the file is opened)  This pointer cannot be NULL.
SQLINTEGER *	<i>IndicatorValue</i>	Input (deferred), output (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.

Table 17. *SQLBindFileToParam* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	<i>FileNameLength</i>	Input (deferred)	Pointer to the location that contains the length of the file name (or an array of lengths) at the time the next <code>SQLExecute()</code> or <code>SQLExecDirect()</code> function is run using the <i>StatementHandle</i> .  If this pointer is <code>NULL</code> , then a length of <code>SQL_NTS</code> is assumed.  The maximum value of the file name length is 255.
SQLSMALLINT	<i>DataType</i>	Input	SQL data type of the column. The data type must be one of: <ul style="list-style-type: none"> <li>• <code>SQL_BLOB</code></li> <li>• <code>SQL_CLOB</code></li> <li>• <code>SQL_DBCLOB</code></li> </ul>
SQLSMALLINT	<i>MaxFileNameLength</i>	Input	This specifies the length of the <i>FileName</i> buffer. If the application calls <code>SQLParamOptions()</code> to specify multiple values for each parameter, this is the length of each element in the <i>FileName</i> array.
SQLSMALLINT	<i>ParameterNumber</i>	Input	Parameter marker number. Parameters are numbered sequentially, from left to right, starting at 1.

## Usage

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

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`

## Error conditions

Table 18. *SQLBindFileToParam* SQLSTATEs

SQLSTATE	Description	Explanation
58004	Unexpected system failure	Unrecoverable system error.
HY004	SQL data type out of range	The value specified for <i>DataType</i> is not a valid SQL type for this function call.
HY009	Argument value that is not valid	<i>FileName</i> , <i>FileOptions</i> <i>FileNameLength</i> , is a null pointer.

Table 18. *SQLBindFileToParam SQLSTATES (continued)*

SQLSTATE	Description	Explanation
HY010	Function sequence error	The function is called while in a data-at-processing (SQLParamData() or SQLPutData()) operation.  The function is called while within a BEGIN COMPOUND and END COMPOUND SQL operation.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value specified for the input argument <i>MaxFileNameLength</i> is less than 0.
HY093	Parameter number that is not valid	The value specified for <i>ParameterNumber</i> is 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.

## References

- “SQLBindParam - Bind a buffer to a parameter marker”
- “SQLExecute - Execute a statement” on page 83
- “SQLParamOptions - Specify an input array for a parameter” on page 163

## SQLBindParam - Bind a buffer to a parameter marker

### Purpose

SQLBindParam() has been deprecated and replaced by SQLBindParameter(). Although this version of DB2 UDB CLI continues to support SQLBindParam(), it is recommended that you begin using SQLBindParameter() in your DB2 UDB CLI programs so that they conform to the latest standards.

SQLBindParam() binds an application variable to a parameter marker in an SQL statement. This function can also be used to bind an application variable to a parameter of a stored procedure CALL statement where the parameter can be input or output.

### Syntax

```
SQLRETURN SQLBindParam (SQLHSTMT    hstmt,
                        SQLSMALLINT ipar,
                        SQLSMALLINT fCType,
                        SQLSMALLINT fSqlType,
                        SQLINTEGER   cbParamDef,
                        SQLSMALLINT ibScale,
                        SQLPOINTER   rgbValue,
                        SQLINTEGER   *pcbValue);
```

### Function arguments

Table 19. *SQLBindParam arguments*

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.

Table 19. SQLBindParam arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	<i>pcbValue</i>	Input (deferred), or output (deferred), or both	<p>A variable whose value is interpreted when the statement is processed:</p> <ul style="list-style-type: none"> <li>• If a null value is used as the parameter, <i>pcbValue</i> must contain the value SQL_NULL_DATA.</li> <li>• If the dynamic argument is supplied at execute-time by calling ParamData() and PutData(), <i>pcbValue</i> must contain the value SQL_DATA_AT_EXEC.</li> <li>• If <i>fcType</i> is SQL_CHAR and the data in <i>rgbValue</i> contains a null-terminated string, <i>pcbValue</i> must either contain the length of the data in <i>rgbValue</i> or contain the value SQL_NTS.</li> <li>• If <i>fcType</i> is SQL_CHAR and the data in <i>rgbValue</i> is not null-terminated, <i>pcbValue</i> must contain the length of the data in <i>rgbValue</i>.</li> <li>• If <i>fcType</i> is an LOB type, <i>pcbValue</i> must contain the length of the data in <i>rgbValue</i>. This length value must be specified in bytes, not the number of double byte characters.</li> <li>• Otherwise, <i>pcbValue</i> must be zero.</li> </ul>
SQLINTEGER	<i>cbParamDef</i>	Input	<p>Precision of the corresponding parameter marker.</p> <ul style="list-style-type: none"> <li>• If <i>fSqlType</i> denotes a single-byte character string (for example, SQL_CHAR), this is the maximum length in bytes sent for this parameter. This length includes the null-termination character.</li> <li>• If <i>fSqlType</i> denotes a double-byte character string (for example, SQL_GRAPHIC), this is the maximum length in double-byte characters for this parameter.</li> <li>• If <i>fSqlType</i> denotes SQL_DECIMAL or SQL_NUMERIC, this is the maximum decimal precision.</li> <li>• Otherwise, this argument is unused.</li> </ul>
SQLPOINTER	<i>rgbValue</i>	Input (deferred) or output (deferred)	<p>At processing time, if <i>pcbValue</i> does not contain SQL_NULL_DATA or SQL_DATA_AT_EXEC, then <i>rgbValue</i> points to a buffer that contains the actual data for the parameter.</p> <p>If <i>pcbValue</i> contains SQL_DATA_AT_EXEC, then <i>rgbValue</i> is an application-defined 32-bit value that is associated with this parameter. This 32-bit value is returned to the application through a later SQLParamData() call.</p>

Table 19. SQLBindParam arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>fCType</i>	Input	<p>Application data type of the parameter. The following types are supported:</p> <ul style="list-style-type: none"> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BLOB</li> <li>• SQL_BLOB_LOCATOR</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_CLOB_LOCATOR</li> <li>• SQL_DATETIME</li> <li>• SQL_DBCLOB</li> <li>• SQL_DBCLOB_LOCATOR</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TYPE_DATE</li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_VARBINARY</li> <li>• SQL_VARCHAR</li> <li>• SQL_VARGRAPHIC</li> <li>• SQL_WCHAR</li> <li>• SQL_WVARCHAR</li> </ul> <p>Specifying SQL_DEFAULT causes data to be transferred from its default application data type to the type indicated in <i>fSqlType</i>.</p>

Table 19. *SQLBindParam* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>fSqlType</i>	Input	SQL data type of the parameter. The supported types are: <ul style="list-style-type: none"> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BLOB</li> <li>• SQL_BLOB_LOCATOR</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_CLOB_LOCATOR</li> <li>• SQL_DATETIME</li> <li>• SQL_DBCLOB</li> <li>• SQL_DBCLOB_LOCATOR</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TYPE_DATE</li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_VARBINARY</li> <li>• SQL_VARCHAR</li> <li>• SQL_VARGRAPHIC</li> <li>• SQL_WCHAR</li> <li>• SQL_WVARCHAR</li> </ul>
SQLSMALLINT	<i>ibScale</i>	Input	Scale of the corresponding parameter if <i>fSqlType</i> is SQL_DECIMAL or SQL_NUMERIC. If <i>fSqlType</i> is SQL_TIMESTAMP, this is the number of digits to the right of the decimal point in the character representation of a timestamp (for example, the scale of yyyy-mm-dd hh:mm:ss.fff is 3).  Other than for the <i>fSqlType</i> values mentioned here, <i>ibScale</i> is unused.
SQLSMALLINT	<i>ipar</i>	Input	Parameter marker number, ordered sequentially left to right, starting at 1.

## Usage

When *SQLBindParam()* is used to bind an application variable to an output parameter for a stored procedure, DB2 UDB CLI provides some performance enhancement if the *rgbValue* buffer is placed consecutively in memory after the *pcbValue* buffer.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 20. SQLBindParam SQLSTATES

SQLSTATE	Description	Explanation
07006	Restricted data type attribute violation	Same as SQLSetParam().
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY003	Program type out of range	Same as SQLSetParam().
HY004	SQL data type out of range	Same as SQLSetParam().
HY009	Argument value that is not valid	Both <i>rgbValue</i> and <i>pcbValue</i> are null pointers, or <i>ipar</i> is less than one.
HY010	Function sequence error	Function is called after SQLExecute() or SQLExecDirect() has returned SQL_NEED_DATA, but data has not been sent for all <i>data-at-execution</i> parameters.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY014	Too many handles	The maximum number of handles has been allocated.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## SQLBindParameter - Bind a parameter marker to a buffer

### Purpose

SQLBindParameter() is used to associate (bind) parameter markers in an SQL statement to application variables. Data is transferred from the application to the Database Management System (DBMS) when SQLExecute() or SQLExecDirect() is called. Data conversion might occur as the data is transferred.

This function must also be used to bind an application storage to a parameter of a stored procedure CALL statement where the parameter can be input, output or both. This function is essentially an extension of SQLSetParam().

### Syntax

```
SQLRETURN SQLBindParameter(SQLHSTMT StatementHandle,
                           SQLSMALLINT ParameterNumber,
                           SQLSMALLINT InputOutputType,
                           SQLSMALLINT ValueType,
                           SQLSMALLINT ParameterType,
                           SQLINTEGER ColumnSize,
                           SQLSMALLINT DecimalDigits,
                           SQLPOINTER ParameterValuePtr,
                           SQLINTEGER BufferLength,
                           SQLINTEGER *StrLen_or_IndPtr);
```



## Function arguments

Table 21. *SQLBindParameter* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLINTEGER	<i>ColumnSize</i>	Input	<p>Precision of the corresponding parameter marker.</p> <ul style="list-style-type: none"><li>• If <i>ParameterType</i> denotes a binary or single-byte character string (for example, <code>SQL_CHAR</code>), this is the maximum length in bytes for this parameter marker.</li><li>• If <i>ParameterType</i> denotes a double-byte character string (for example, <code>SQL_GRAPHIC</code>), this is the maximum length in double-byte characters for this parameter.</li><li>• If <i>ParameterType</i> denotes <code>SQL_DECIMAL</code> or <code>SQL_NUMERIC</code>, this is the maximum decimal precision.</li><li>• Otherwise, this argument is ignored.</li></ul>

Table 21. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER *	<i>StrLen_or_IndPtr</i>	Input (deferred), output (deferred)	<p>If this is an input or input/output parameter, this is the pointer to the location that contains (when the statement is processed) the length of the parameter marker value stored at <i>ParameterValuePtr</i>.</p> <p>To specify a null value for a parameter marker, this storage location must contain SQL_NULL_DATA.</p> <p>If <i>ValueType</i> is SQL_C_CHAR, this storage location must contain either the exact length of the data stored at <i>ParameterValuePtr</i>, or SQL_NTS if the content at <i>ParameterValuePtr</i> is null-terminated.</p> <p>If <i>ValueType</i> indicates LOB data, this storage location must contain the length of the data stored at <i>ParameterValuePtr</i>. This length value must be specified in bytes, not the number of double-byte characters.</p> <p>If <i>ValueType</i> indicates character data (explicitly, or implicitly using SQL_C_DEFAULT), and this pointer is set to NULL, it is assumed that the application always provides a null-terminated string in <i>ParameterValuePtr</i>. This also implies that this parameter marker never has a null value.</p> <p>If <i>ValueType</i> specifies any form of double-byte character data, then <i>StrLen_or_IndPtr</i> must be the number of double-byte characters, not the number of bytes.</p> <p>When <i>SQLExecute()</i> or <i>SQLExecDirect()</i> is called, and <i>StrLen_or_IndPtr</i> points to a value of SQL_DATA_AT_EXEC, the data for the parameter is sent with <i>SQLPutData()</i>. This parameter is referred to as a <i>data-at-execution</i> parameter.</p>
SQLINTEGER	<i>BufferLength</i>	Input	Not used.

Table 21. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLPOINTER	<i>ParameterValuePtr</i>	Input (deferred), or output (deferred), or both	<ul style="list-style-type: none"> <li>• On input (<i>InputOutputType</i> set to SQL_PARAM_INPUT, or SQL_PARAM_INPUT_OUTPUT), the following situations are true:               <p>At processing time, if <i>StrLen_or_IndPtr</i> does not contain SQL_NULL_DATA or SQL_DATA_AT_EXEC, then <i>ParameterValuePtr</i> points to a buffer that contains the actual data for the parameter.</p> <p>If <i>StrLen_or_IndPtr</i> contains SQL_DATA_AT_EXEC, then <i>ParameterValuePtr</i> 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.</p> <p>If SQLParamOptions() is called to specify multiple values for the parameter, then <i>ParameterValuePtr</i> is a pointer to an input buffer array of <i>BufferLength</i> bytes.</p> </li> <li>• On output (<i>InputOutputType</i> set to SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT), the following situations are true:               <p><i>ParameterValuePtr</i> points to the buffer where the output parameter value of the stored procedure is stored.</p> <p>If <i>InputOutputType</i> is set to SQL_PARAM_OUTPUT, and both <i>ParameterValuePtr</i> and <i>StrLen_or_IndPtr</i> are NULL pointers, then the output parameter value or the return value from the stored procedure call is discarded.</p> </li> </ul>
SQLSMALLINT	<i>DecimalDigits</i>	Input	<p>Scale of the corresponding parameter if <i>ParameterType</i> is SQL_DECIMAL or SQL_NUMERIC. If <i>ParameterType</i> is SQL_TYPE_TIMESTAMP, this is the number of digits to the right of the decimal point in the character representation of a timestamp (for example, the scale of yyyy-mm-dd hh:mm:ss.fff is 3).</p> <p>Other than for the <i>ParameterType</i> values mentioned here, <i>DecimalDigits</i> is ignored.</p>

Table 21. SQLBindParameter arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>InputOutputType</i>	Input	<p>The type of parameter. The value of the SQL_DESC_PARAMETER_TYPE field of the implementation parameter descriptor (IPD) is also set to this argument. The supported types are:</p> <ul style="list-style-type: none"> <li>• <b>SQL_PARAM_INPUT:</b> 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 processed, the actual data value for the parameter is sent to the server: the <i>ParameterValuePtr</i> buffer must contain valid input data values; the <i>StrLen_or_IndPtr</i> buffer must contain the corresponding length value or SQL_NTS, SQL_NULL_DATA, or (if the value should be sent via SQLParamData() and SQLPutData()) SQL_DATA_AT_EXEC.</li> <li>• <b>SQL_PARAM_INPUT_OUTPUT:</b> The parameter marker is associated with an input/output parameter of the CALLED stored procedure. When the statement is processed, actual data value for the parameter is sent to the server: the <i>ParameterValuePtr</i> buffer must contain valid input data values; the <i>StrLen_or_IndPtr</i> buffer must contain the corresponding length value or SQL_NTS, SQL_NULL_DATA, or (if the value should be sent via SQLParamData() and SQLPutData()) SQL_DATA_AT_EXEC.</li> <li>• <b>SQL_PARAM_OUTPUT:</b> 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 processed, data for the output parameter is returned to the application buffer specified by <i>ParameterValuePtr</i> and <i>StrLen_or_IndPtr</i>, unless both are NULL pointers, in which case the output data is discarded. If an output parameter does not have a return value then <i>StrLen_or_IndPtr</i> is set to SQL_NULL_DATA.</li> </ul>
SQLSMALLINT	<i>ParameterNumber</i>	Input	Parameter marker number, ordered sequentially left to right, starting at 1.
SQLSMALLINT	<i>ParameterType</i>	Input	SQL data type of the parameter.

Table 21. *SQLBindParameter* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>ValueType</i>	Input	<p>C data type of the parameter. The following types are supported:</p> <ul style="list-style-type: none"> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BLOB</li> <li>• SQL_BLOB_LOCATOR</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_CLOB_LOCATOR</li> <li>• SQL_DATETIME</li> <li>• SQL_DBCLOB</li> <li>• SQL_DBCLOB_LOCATOR</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TYPE_DATE</li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_VARBINARY</li> <li>• SQL_VARCHAR</li> <li>• SQL_VARGRAPHIC</li> <li>• SQL_WCHAR</li> <li>• SQL_WCHAR</li> <li>• SQL_WVARCHAR</li> <li>• SQL_WVARCHAR</li> </ul> <p>Specifying SQL_C_DEFAULT causes data to be transferred from its default C data type to the type indicated in <i>ParameterType</i>.</p>

## Usage

A parameter marker is represented by a "?" character in an SQL statement and is used to indicate a position in the statement where an application supplied value is to be substituted when the statement is processed. This value is obtained from an application variable.

The application must bind a variable to each parameter marker in the SQL statement before executing the SQL statement. For this function, *ParameterValuePtr* and *StrLen\_or\_IndPtr* are deferred arguments; the storage locations must be valid and contain input data values when the statement is processed. 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 declared statically or globally.

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

All parameters bound by this function remain in effect until `SQLFreeStmt()` is called with either the `SQL_DROP` or `SQL_RESET_PARAMS` option, or until `SQLBindParameter()` is called again for the same parameter *ParameterNumber* number.

After the SQL statement and the results have been processed, the application might want to reuse the statement handle to process a different SQL statement. If the parameter marker specifications are different (number of parameters, length or type), then `SQLFreeStmt()` should be called with `SQL_RESET_PARAMS` to reset or clear the parameter bindings.

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

Because the data in the variables referenced by *ParameterValuePtr* and *StrLen\_or\_IndPtr* is not verified until the statement is processed, data content or format errors are not detected or reported until `SQLExecute()` or `SQLExecDirect()` is called.

`SQLBindParameter()` essentially extends the capability of the `SQLSetParam()` function by providing a method of specifying whether a parameter is input, input and output, or output. This information is necessary for the proper handling of parameters for stored procedures.

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. Also, note that these types should be consistent with the parameter types specified when the stored procedure is registered with the SQL `CREATE PROCEDURE` statement.

- 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 UDB 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 UDB 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 UDB CLI discards the output value. If the data source does not return a value for an output parameter, DB2 UDB CLI sets the *StrLen\_or\_IndPtr* buffer to `SQL_NULL_DATA`.
- For this function, both *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 processed. 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 processed.

When `SQLBindParameter()` is used to bind an application variable to an output parameter for a stored procedure, DB2 UDB 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;

```

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Error conditions

Table 22. SQLBindParameter SQLSTATES

SQLSTATE	Description	Explanation
07006	Conversion not valid	The conversion from the data value identified by the <i>ValueType</i> argument to the data type identified by the <i>ParameterType</i> argument is not a meaningful conversion. (For example, conversion from SQL_C_DATE to SQL_DOUBLE.)
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
58004	Unexpected system failure	Unrecoverable system error.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY003	Program type out of range	The value specified by the argument <i>ParameterNumber</i> not a valid data type or SQL_C_DEFAULT.
HY004	SQL data type out of range	The value specified for the argument <i>ParameterType</i> is not a valid SQL data type.
HY009	Argument value not valid	The argument <i>ParameterValuePtr</i> is a null pointer and the argument <i>StrLen_or_IndPtr</i> is a null pointer, and <i>InputOutputType</i> is not SQL_PARAM_OUTPUT.
HY010	Function sequence error	Function is called after SQLExecute() or SQLExecDirect() has returned SQL_NEED_DATA, but data has not been sent for all <i>data-at-execution</i> parameters.
HY013	Unexpected memory handling error	DB2 UDB CLI is unable to access memory required to support the processing or completion of the function.
HY014	Too many handles	The maximum number of handles has been allocated.
HY021	Inconsistent descriptor information	The descriptor information checked during a consistency check is not consistent.
HY090	String or buffer length not valid	The value specified for the argument <i>BufferLength</i> is less than 0.
HY093	Parameter number not valid	The value specified for the argument <i>ValueType</i> is less than 1 or greater than the maximum number of parameters supported by the server.
HY094	Scale value not valid	The value specified for <i>ParameterType</i> is either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>DecimalDigits</i> is less than 0 or greater than the value for the argument <i>ParamDef</i> (precision).  The value specified for <i>ParameterType</i> is SQL_C_TIMESTAMP and the value for <i>ParameterType</i> is either SQL_CHAR or SQL_VARCHAR and the value for <i>DecimalDigits</i> is less than 0 or greater than 6.

Table 22. *SQLBindParameter SQLSTATES (continued)*

SQLSTATE	Description	Explanation
HY104	Precision value not valid	The value specified for <i>ParameterType</i> is either SQL_DECIMAL or SQL_NUMERIC and the value specified for <i>ParamDef</i> is less than 1.
HY105	Parameter type not valid	<i>InputOutputType</i> is not one of SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT.
HYC00	Driver not capable	DB2 UDB CLI or data source does not support the conversion specified by the combination of the value specified for the argument <i>ValueType</i> and the value specified for the argument <i>ParameterType</i> .  The value specified for the argument <i>ParameterType</i> is not supported by either DB2 UDB CLI or the data source.

## References

- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLParamData - Get next parameter for which a data value is needed” on page 162
- “SQLPutData - Pass data value for a parameter” on page 179

## SQLCancel - Cancel statement

### Purpose

SQLCancel() attempts to end the processing of an ongoing SQL statement operation that is executing synchronously. To cancel the function, the application calls SQLCancel() with the same statement handle that is used by the target function, but on a different thread. How the function is canceled depends on the operating system.

### Syntax

```
SQLRETURN SQLCancel (SQLHSTMT hstmt);
```

### Function arguments

Table 23. *SQLCancel arguments*

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle

### Usage

A successful return code indicates that the implementation has accepted the cancel request; it does not ensure that the processing is cancelled.

### Return codes

- SQL\_SUCCESS
- SQL\_INVALID\_HANDLE
- SQL\_ERROR



## Diagnostics

Table 24. *SQLCancel SQLSTATEs*

SQLSTATE	Description	Explanation
HY009 *	Argument value that is not valid	<i>hstmt</i> is not a statement handle.

## Restrictions

DB2 UDB CLI does not support asynchronous statement processing.

## SQLCloseCursor - Close cursor statement

### Purpose

`SQLCloseCursor()` closes the open cursor on a statement handle.

### Syntax

```
SQLRETURN SQLCloseCursor (SQLHSTMT hstmt);
```

### Function arguments

Table 25. *SQLCloseCursor arguments*

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle

## Usage

Calling `SQLCloseCursor()` closes any cursor associated with the statement handle and discards any pending results. If no open cursor is associated with the statement handle, the function has no effect.

If the statement handle references a stored procedure that has multiple result sets, the `SQLCloseCursor()` closes only the current result set. Any additional result sets remain open and usable.

## Return codes

- `SQL_SUCCESS`
- `SQL_INVALID_HANDLE`
- `SQL_ERROR`

## Diagnostics

Table 26. *SQLCloseCursor SQLSTATEs*

SQLSTATE	Description	Explanation
08003 *	Connection not open	The connection for <i>hstmt</i> is not established.
HY009 *	Argument value that is not valid	<i>hstmt</i> is not a statement handle.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## SQLColAttributes - Obtain column attributes

### Purpose

SQLColAttributes() obtains an attribute for a column of the result set, and is also used to determine the number of columns. SQLColAttributes() is a more extensible alternative to the SQLDescribeCol() function.

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

This function (or SQLDescribeCol()) must be called before SQLBindCol(), if the application does not know the various attributes (such as data type and length) of the column.

### Syntax

```
SQLRETURN SQLColAttributes (SQLHSTMT      hstmt,
                           SQLSMALLINT   icol,
                           SQLSMALLINT   fDescType,
                           SQLCHAR       *rgbDesc,
                           SQLINTEGER    cbDescMax,
                           SQLINTEGER    *pcbDesc,
                           SQLINTEGER    *pfDesc);
```

### Function arguments

Table 27. SQLColAttributes arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>rgbDesc</i>	Output	Pointer to buffer for string column attributes.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER *	<i>pcbDesc</i>	Output	Actual number of bytes in the descriptor to return. If this argument contains a value equal to or higher than the length <i>rgbDesc</i> buffer, truncation has occurred. The descriptor is then truncated to <i>cbDescMax</i> - 1 bytes.
SQLINTEGER *	<i>pfDesc</i>	Output	Pointer to integer which holds information regarding numeric column attributes.
SQLINTEGER	<i>cbDescMax</i>	Input	Length of descriptor buffer ( <i>rgbDesc</i> )
SQLSMALLINT	<i>fDescType</i>	Input	Supported values are described in Table 28.
SQLSMALLINT	<i>icol</i>	Input	Column number in result set (must be between 1 and the number of columns in the results set inclusive). This argument is ignored when SQL_DESC_COUNT is specified.

Table 28. fDescType descriptor types

Descriptor	Type	Description
SQL_DESC_AUTO_INCREMENT	INTEGER	This is SQL_TRUE if the column can be incremented automatically upon insertion of a new row to the table. SQL_FALSE if the column cannot be incremented automatically.

Table 28. *fDescType* descriptor types (continued)

Descriptor	Type	Description
SQL_DESC_BASE_COLUMN	CHAR(128)	<p>The name of the actual column in the underlying table over which this column is built.</p> <p>For this attribute to be retrieved, the attribute SQL_ATTR_EXTENDED_COL_INFO must have been set to SQL_TRUE for either the statement handle or the connection handle.</p>
SQL_DESC_BASE_SCHEMA	CHAR(128)	<p>The schema name of the underlying table over which this column is built.</p> <p>For this attribute to be retrieved, the attribute SQL_ATTR_EXTENDED_COL_INFO must have been set to SQL_TRUE for either the statement handle or the connection handle.</p>
SQL_DESC_BASE_TABLE	CHAR(128)	<p>The name of the underlying table over which this column is built.</p> <p>For this attribute to be retrieved, the attribute SQL_ATTR_EXTENDED_COL_INFO must have been set to SQL_TRUE for either the statement handle or the connection handle.</p>
SQL_DESC_COUNT	SMALLINT	The number of columns in the result set is returned in <i>pfDesc</i> .
SQL_DESC_DISPLAY_SIZE	SMALLINT	The maximum number of bytes needed to display the data in character form is returned in <i>pfDesc</i> .
SQL_DESC_LABEL	CHAR(128)	<p>The label for this column, if one exists. Otherwise, a zero-length string.</p> <p>For this attribute to be retrieved, the attribute SQL_ATTR_EXTENDED_COL_INFO must have been set to SQL_TRUE for either the statement handle or the connection handle.</p>
SQL_DESC_LENGTH	INTEGER	<p>The number of <i>bytes</i> of data associated with the column is returned in <i>pfDesc</i>.</p> <p>If the column identified in <i>icol</i> is character based, for example, SQL_CHAR, SQL_VARCHAR, or SQL_LONG_VARCHAR, the actual length or maximum length is returned.</p> <p>If the column type is SQL_DECIMAL or SQL_NUMERIC, SQL_DESC_LENGTH is (<i>precision</i> * 256) + <i>scale</i>. This is returned so that the same value can be passed as input on SQLBindCol(). The precision and scale can also be obtained as separate values for these data types by using SQL_DESC_PRECISION and SQL_DESC_SCALE.</p>
SQL_DESC_NAME	CHAR(128)	The name of the column <i>icol</i> is returned in <i>rgbDesc</i> . If the column is an expression, then the result returned is product specific.
SQL_DESC_NULLABLE	SMALLINT	<p>If the column identified by <i>icol</i> can contain nulls, then SQL_NULLABLE is returned in <i>pfDesc</i>.</p> <p>If the column is constrained not to accept nulls, then SQL_NO_NULLS is returned in <i>pfDesc</i>.</p>

Table 28. *fDescType* descriptor types (continued)

Descriptor	Type	Description
SQL_DESC_PRECISION	SMALLINT	The precision attribute of the column is returned.
SQL_DESC_SCALE	SMALLINT	The scale attribute of the column is returned.
SQL_DESC_SEARCHABLE	INTEGER	<p>This is SQL_UNSEARCHABLE if the column cannot be used in a <b>WHERE</b> clause.</p> <p>This is SQL_LIKE_ONLY if the column can be used in a <b>WHERE</b> clause only with the <b>LIKE</b> predicate.</p> <p>This is SQL_ALL_EXCEPT_LIKE if the column can be used in a <b>WHERE</b> clause with all comparison operators except <b>LIKE</b>.</p> <p>This is SQL_SEARCHABLE if the column can be used in a <b>WHERE</b> clause with any comparison operator.</p> <p>For this attribute to be retrieved, the attribute SQL_ATTR_EXTENDED_COL_INFO must have been set to SQL_TRUE for either the statement handle or the connection handle.</p>
SQL_DESC_TYPE_NAME	CHAR(128)	The character representation of the SQL data type of the column identified in <i>icol</i> . This is returned in <i>rgbDesc</i> . The possible values for the SQL data type are listed in Table 3 on page 16. In addition, user-defined type (UDT) information is also returned. The format for the UDT is <schema name qualifier><job's current separator><UDT name>.
SQL_DESC_TYPE	SMALLINT	The SQL data type of the column identified in <i>icol</i> is returned in <i>pfDesc</i> . The possible values for <i>pfSqlType</i> are listed in Table 3 on page 16.
SQL_DESC_UNNAMED	SMALLINT	This is SQL_NAMED if the NAME field is an actual name, or SQL_UNNAMED if the NAME field is an implementation-generated name.
SQL_DESC_UPDATABLE	INTEGER	<p>Column is described by the values for the defined constants:</p> <p>SQL_ATTR_READONLY  SQL_ATTR_WRITE  SQL_ATTR_READWRITE_UNKNOWN</p> <p>SQL_COLUMN_UPDATABLE describes the updatability of the column in the result set. Whether a column can be updated can be based on the data type, user privileges, and the definition of the result set itself. If it is unclear whether a column can be updated, SQL_ATTR_READWRITE_UNKNOWN should be returned.</p> <p>For this attribute to be retrieved, the attribute SQL_ATTR_EXTENDED_COL_INFO must have been set to SQL_TRUE for either the statement handle or the connection handle.</p>

## Usage

Instead of returning a specific set of arguments like `SQLDescribeCol()`, `SQLColAttributes()` can be used to specify which attribute you want to receive for a specific column. If the required information is a string, it is returned in `rgbDesc`. If the required information is a number, it is returned in `pfDesc`.

Although `SQLColAttributes()` allows for future extensions, it requires more calls to receive the same information than `SQLDescribeCol()` for each column.

If a `fDescType` descriptor type does not apply to the database server, an empty string is returned in `rgbDesc` or zero is returned in `pfDesc`, depending on the expected result of the descriptor.

Columns are identified by a number (numbered sequentially from left to right starting with 1) and can be described in any order.

Calling `SQLColAttributes()` with `fDescType` set to `SQL_DESC_COUNT` is an alternative to calling `SQLNumResultCols()` to determine whether any columns can be returned.

Call `SQLNumResultCols()` before calling `SQLColAttributes()` to determine whether a result set exists.

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

## Diagnostics

Table 29. `SQLColAttributes` `SQLSTATEs`

SQLSTATE	Description	Explanation
07009	Column number that is not valid	The value specified for the argument <code>icol</code> is less than 1.
HY009	Argument value that is not valid	The value specified for the argument <code>fDescType</code> is not equal to a value specified in Table 28 on page 52.  The argument <code>rgbDesc</code> , <code>pcbDesc</code> , or <code>pfDesc</code> is a null pointer.
HY010	Function sequence error	The function is called before calling <code>SQLPrepare()</code> or <code>SQLExecDirect()</code> for the <code>hstmt</code> .
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYC00	Driver not capable	The SQL data type returned by the database server for column <code>icol</code> is not recognized by DB2 UDB CLI.

## References

- “`SQLBindCol` - Bind a column to an application variable” on page 29
- “`SQLDescribeCol` - Describe column attributes” on page 68
- “`SQLExecDirect` - Execute a statement directly” on page 81
- “`SQLExecute` - Execute a statement” on page 83
- “`SQLPrepare` - Prepare a statement” on page 164

# SQLColumnPrivileges - Get privileges associated with the columns of a table

## Purpose

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.

## Syntax

```
SQLRETURN SQLColumnPrivileges (
    SQLHSTMT          StatementHandle,
    SQLCHAR           *CatalogName,
    SQLSMALLINT      NameLength1,
    SQLCHAR           *SchemaName,
    SQLSMALLINT      NameLength2,
    SQLCHAR           *TableName,
    SQLSMALLINT      NameLength3,
    SQLCHAR           *ColumnName,
    SQLSMALLINT      NameLength4);
```

## Function arguments

Table 30. SQLColumnPrivileges arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>Statement Handle</i>	Input	Statement handle.
SQLCHAR *	<i>CatalogName</i>	Input	Catalog qualifier of a 3 part table name. This must be a NULL pointer or a zero length string.
SQLSMALLINT	<i>NameLength1</i>	Input	Length of <i>CatalogName</i> . This must be set to 0.
SQLCHAR *	<i>SchemaName</i>	Input	Schema qualifier of table name.
SQLSMALLINT	<i>NameLength2</i>	Input	Length of <i>SchemaName</i> .
SQLCHAR *	<i>TableName</i>	Input	Table Name.
SQLSMALLINT	<i>NameLength3</i>	Input	Length of <i>TableName</i> .
SQLCHAR *	<i>ColumnName</i>	Input	Buffer that can contain a <i>pattern-value</i> to qualify the result set by column name.
SQLSMALLINT	<i>NameLength4</i>	Input	Length of <i>ColumnName</i> .

## Usage

The results are returned as a standard result set containing the columns listed in Table 31 on page 57. 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\_SCHEM, TABLE\_NAME, COLUMN\_NAME columns of the SQLColumns() result set as input arguments to this function

Because 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.

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. Because DB2 names are less than 128, the application can choose to always set aside 128 characters (plus the null-terminator) for the output buffer, or alternatively, call SQLGetInfo() with SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN. The SQL\_MAX\_CATALOG\_NAME\_LEN value determines the actual length of the TABLE\_CAT supported by the connected Database Management System (DBMS). The SQL\_MAX\_SCHEMA\_NAME\_LEN value determines the actual length of the TABLE\_SCHEM supported by the connected DBMS. The SQL\_MAX\_TABLE\_NAME\_LEN value determines the actual length of the TABLE\_NAME supported by the connected DBMS. The SQL\_MAX\_COLUMN\_NAME\_LEN value determines the actual length of the COLUMN\_NAME supported by the connected DBMS.

Note that the *ColumnName* argument accepts a search pattern.

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

Table 31. Columns returned by SQLColumnPrivileges

Column number/name	Data type	Description
COLUMN_NAME	VARCHAR(128) not NULL	Name of the column of the specified table or view.
GRANTEE	VARCHAR(128)	Authorization ID of the user to whom the privilege is granted.
GRANTOR	VARCHAR(128)	Authorization ID of the user who granted the privilege.
IS_GRANTABLE	VARCHAR(3)	Indicates whether the grantee is permitted to grant the privilege to other users.  Either YES or NO.
PRIVILEGE	VARCHAR(128)	The column privilege. This can be: <ul style="list-style-type: none"> <li>• INSERT</li> <li>• REFERENCES</li> <li>• SELECT</li> <li>• UPDATE</li> </ul>
TABLE_CAT	VARCHAR(128)	This is always NULL.
TABLE_NAME	VARCHAR(128) not NULL	Name of the table or view.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing TABLE_NAME.

**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 32. SQLColumnPrivileges SQLSTATES

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal to SQL_NTS.
HY010	Function sequence error	There is an open cursor for this statement handle, or there is no connection for this statement handle.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## Restrictions

None

## Example

```
/* From the CLI sample TBINFO.C */
/* ... */

/* call SQLColumnPrivileges */
printf("\n Call SQLColumnPrivileges for:\n");
printf(" tbSchema = %s\n", tbSchema);
printf(" tbName = %s\n", tbName);
sqlrc = SQLColumnPrivileges( hstmt, NULL, 0,
                             tbSchema, SQL_NTS,
                             tbName, SQL_NTS,
                             colNamePattern, SQL_NTS);
```

## References

- "SQLColumns - Get column information for a table"
- "SQLTables - Get table information" on page 215

## SQLColumns - Get column information for a table

### Purpose

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 SELECT statement.

### Syntax

```
SQLRETURN SQLColumns (SQLHSTMT    hstmt,
                     SQLCHAR      *szCatalogName,
                     SQLSMALLINT  cbCatalogName,
                     SQLCHAR      *szSchemaName,
                     SQLSMALLINT  cbSchemaName,
```



```

SQLCHAR      *szTableName,
SQLSMALLINT  cbTableName,
SQLCHAR      *szColumnName,
SQLSMALLINT  cbColumnName);

```

## Function arguments

Table 33. *SQLColumns* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLCHAR *	<i>szCatalogName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set. <i>Catalog</i> is the first part of a three-part table name.  This must be a NULL pointer or a zero length string.
SQLSMALLINT	<i>cbCatalogName</i>	Input	Length of <i>szCatalogName</i> . This must be set to 0.
SQLCHAR *	<i>szSchemaName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by schema name.
SQLSMALLINT	<i>cbSchemaName</i>	Input	Length of <i>szSchemaName</i>
SQLCHAR *	<i>szTableName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by table name.
SQLSMALLINT	<i>cbTableName</i>	Input	Length of <i>szTableName</i>
SQLCHAR *	<i>szColumnName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by column name.
SQLSMALLINT	<i>cbColumnName</i>	Input	Length of <i>szColumnName</i>

## Usage

This function retrieves information about the columns of a table or a list of tables.

`SQLColumns()` returns a standard result set. Table 34 lists the columns in the result set. Applications should anticipate that additional columns beyond the REMARKS columns can be added in future releases.

The *szCatalogName*, *szSchemaName*, *szTableName*, and *szColumnName* arguments accept search patterns. An escape character can be specified in conjunction with a wildcard character to allow that actual character to be used in the search pattern. The escape character is specified on the `SQL_ATTR_ESCAPE_CHAR` environment attribute.

This function does not return information about the columns in a result set, which is retrieved by `SQLDescribeCol()` or `SQLColAttributes()`. If an application wants to obtain column information for a result set, it should always call `SQLDescribeCol()` or `SQLColAttributes()` for efficiency. `SQLColumns()` maps to a complex query against the system catalogs, and can require a large amount of system resources.

Table 34. *Columns returned by SQLColumns*

Column name	Data type	Description
BUFFER_LENGTH	INTEGER	The maximum number of bytes to store data from this column if <code>SQL_DEFAULT</code> were specified on the <code>SQLBindCol()</code> , <code>SQLGetData()</code> and <code>SQLBindParam()</code> calls.

Table 34. Columns returned by SQLColumns (continued)

Column name	Data type	Description
CHAR_OCTET_LENGTH	INTEGER	Contains the maximum length in octets for a character data type column. For single byte character sets, this is the same as LENGTH_PRECISION. For all other data types, it is NULL.
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 is a <i>pseudo-literal</i> , 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 is 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 is specified, then this column is NULL.
COLUMN_NAME	VARCHAR(128)	Column identifier. The name of the column of the specified view, table, or table's column the alias is built for.
DATA_TYPE	SMALLINT not NULL	DATA_TYPE identifies the SQL data type of the column. For CHAR FOR BIT DATA and VARCHAR FOR BIT DATA data types, the CLI returns SQL_BINARY and SQL_VARBINARY to indicate it is a FOR BIT DATA column.
DATETIME_CODE	INTEGER	The subtype code for date and time data types: <ul style="list-style-type: none"> <li>• SQL_DATE</li> <li>• SQL_TIME</li> <li>• SQL_TIMESTAMP</li> </ul> For all other data types, this column returns NULL.
LENGTH_PRECISION	INTEGER	If DATA_TYPE is an approximate numeric data type, this column contains the number of bits of mantissa precision of the column. For exact numeric data types, this column contains the total number of decimal digit allowed in the column. For time and timestamp data types, this column contains the number of digits of precision of the fractional seconds component; otherwise, this column is NULL. <b>Note:</b> The ODBC definition of precision is typically the number of digits to store the data type.

Table 34. Columns returned by SQLColumns (continued)

Column name	Data type	Description
NULLABLE	SMALLINT not NULL	SQL_NO_NULLS if the column does not accept NULL values.  SQL_NULLABLE if the column accepts NULL values.
NUM_PREC_RADIX	SMALLINT	The value is 10, 2, or NULL. If DATA_TYPE is an approximate numeric data type, this column contains the value 2; then the LENGTH_PRECISION 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 LENGTH_PRECISION and NUM_SCALE columns contain the number of decimal digits allowed for the column.  For numeric data types, the Database Management System (DBMS) can return a NUM_PREC_RADIX of either 10 or 2.  NULL is returned for data types where radix is not applicable.
NUM_SCALE	SMALLINT	The scale of the column. NULL is returned for data types where scale is not applicable.
ORDINAL_POSITION	INTEGER NOT NULL	The ordinal position of the column in the table. The first column in the table is number 1.
REMARKS	VARCHAR(254)	Might contain descriptive information about the column.
TABLE_CAT	VARCHAR(128)	The current server.
TABLE_NAME	VARCHAR(128)	Name of the table, view or alias.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing TABLE_NAME.
TYPE_NAME	VARCHAR(128) not NULL	TYPE_NAME is a character string representing the name of the data type corresponding to DATA_TYPE. If the data type is FOR BIT DATA, then the corresponding string FOR BIT DATA is appended to the data type, for example, CHAR () FOR BIT DATA.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 35. SQLColumns SQLSTATES

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal SQL_NTS.
HY010	Function sequence error	There is an open cursor for this statement handle, or there is no connection for this statement handle.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## SQLConnect - Connect to a data source

### Purpose

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

SQLAllocConnect() must be called before calling this function.

This function must be called before calling SQLAllocStmt().

### Syntax

```
SQLRETURN SQLConnect (SQLHDBC          hdbc,  
                     SQLCHAR          *szDSN,  
                     SQLSMALLINT      cbDSN,  
                     SQLCHAR          *szUID,  
                     SQLSMALLINT      cbUID,  
                     SQLCHAR          *szAuthStr,  
                     SQLSMALLINT      cbAuthStr);
```

### Function arguments

Table 36. SQLConnect arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szAuthStr</i>	Input	Authentication string (password).
SQLCHAR *	<i>szDSN</i>	Input	Data source: name or alias name of the database.
SQLCHAR *	<i>szUID</i>	Input	Authorization name (user identifier).
SQLHDBC	<i>hdbc</i>	Input	Connection handle.
SQLSMALLINT	<i>cbAuthStr</i>	Input	Length of contents of <i>szAuthStr</i> argument.
SQLSMALLINT	<i>cbDSN</i>	Input	Length of contents of <i>szDSN</i> argument.
SQLSMALLINT	<i>cbUID</i>	Input	Length of contents of <i>szUID</i> argument.


## Usage

You can define various connection characteristics (options) in the application using `SQLSetConnectOption()`.

The input length arguments to `SQLConnect()` (*cbDSN*, *cbUID*, *cbAuthStr*) can be set to the actual length of their associated data. This does not include any null-terminating character or to `SQL_NTS` to indicate that the associated data is null-terminated.

Leading and trailing blanks in the *szDSN* and *szUID* argument values are stripped before processing unless they are enclosed in quotation marks.

When running in server mode, both *szUID* and *szAuthStr* must be passed in order for the connection to run on behalf of a user ID other than the current user. If either parameter is `NULL` or both are `NULL`, the connection is started using the user ID that is in effect for the current job running the CLI program.

The data source must already be defined on the system for the connect function to work. On the IBM  platform, you can use the Work with Relational Database Directory Entries (`WRKRDBDIRE`) command to determine which data sources have been defined already, and to optionally define additional data sources.

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

## Diagnostics

Table 37. *SQLConnect* *SQLSTATEs*

<b>SQLSTATE</b>	<b>Description</b>	<b>Explanation</b>
08001	Unable to connect to data source	The driver is unable to establish a connection with the data source (server).
08002	Connection in use	The specified <i>hdbc</i> has been used to establish a connection with a data source and the connection is still open.
08004	Data source rejected establishment of connection	The data source (server) rejected the establishment of the connection.
28000	Authorization specification that is not valid	The value specified for the argument <i>szUID</i> or the value specified for the argument <i>szAuthStr</i> violated restrictions defined by the data source.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.

Table 37. SQLConnect SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	<p>The value specified for argument <i>cbDSN</i> is less than 0, but not equal to SQL_NTS and the argument <i>szDSN</i> is not a null pointer.</p> <p>The value specified for argument <i>cbUID</i> is less than 0, but not equal to SQL_NTS and the argument <i>szUID</i> is not a null pointer.</p> <p>The value specified for argument <i>cbAuthStr</i> is less than 0, but not equal to SQL_NTS and the argument <i>szAuthStr</i> is not a null pointer.</p> <p>A nonmatching double quotation mark (") is found in either the <i>szDSN</i>, <i>szUID</i>, or <i>szAuthStr</i> argument.</p>
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY501 *	Data source name that is not valid	A data source name that is not valid is specified in argument <i>szDSN</i> .

## Restrictions

The implicit connection (or default database) option for IBM DBMSs is not supported. SQLConnect() must be called before any SQL statements can be processed. i5/OS does not support multiple simultaneous connections to the same data source in a single job.

When you are using DB2 UDB CLI on a newer release, SQLConnect() can encounter an SQL0144 message. This indicates that the data source (the server) has obsolete SQL packages that must be deleted. To delete these packages, run the following command on the server system:

```
DLTSQLPKG SQLPKG(QGPL/QSQCLI*)
```

The next SQLConnect() creates a new SQL package.

## Example

Refer to the example in SQLAllocEnv().

## References

- "SQLAllocConnect - Allocate connection handle" on page 21
- "SQLAllocStmt - Allocate a statement handle" on page 27

## SQLCopyDesc - Copy description statement

### Purpose

SQLCopyDesc() copies the fields of the data structure associated with the source handle to the data structure associated with the target handle.

Any existing data in the data structure associated with the target handle is overwritten, except that the ALLOC\_TYPE field is not changed.

### Syntax

```
SQLRETURN SQLCopyDesc (SQLHDESC    sDesc)
                    (SQLHDESC    tDesc);
```

## Function arguments

Table 38. SQLCancel arguments

Data type	Argument	Use	Description
SQLHDESC	<i>sDesc</i>	Input	Source descriptor handle
SQLHDESC	<i>tDesc</i>	Input	Target descriptor handle

## Usage

Handles for the automatically-generated row and parameter descriptors of a statement can be obtained by calling `GetStmtAttr()`.

## Return codes

- SQL\_SUCCESS
- SQL\_INVALID\_HANDLE
- SQL\_ERROR

## SQLDataSources - Get list of data sources

### Purpose

`SQLDataSources()` returns a list of target databases available, one at a time. A database must be cataloged to be available. For more information about cataloging, refer to the usage notes for `SQLConnect()` or see the online help for the Work with Relational Database (RDB) Directory Entries (WRKRDBDIRE) command.

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

If you are running DB2 UDB CLI in SQL server mode, some restrictions apply when you use `SQLDataSources()`.

### Syntax

```
SQLRETURN  SQLDataSources (SQLHENV      EnvironmentHandle,
                          SQLSMALLINT  Direction,
                          SQLCHAR       *ServerName,
                          SQLSMALLINT  BufferLength1,
                          SQLSMALLINT  *NameLength1Ptr,
                          SQLCHAR       *Description,
                          SQLSMALLINT  BufferLength2,
                          SQLSMALLINT  *NameLength2Ptr);
```

## Function arguments

Table 39. SQLDataSources arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>Description</i>	Output	Pointer to buffer where the description of the data source is returned. DB2 UDB CLI returns the <b>Comment</b> field associated with the database catalogued to the Database Management System (DBMS).
SQLCHAR *	<i>ServerName</i>	Output	Pointer to buffer to hold the data source name retrieved.
SQLHENV	<i>EnvironmentHandle</i>	Input	Environment handle.

Table 39. *SQLDataSources* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>Direction</i>	Input	Used by application to request the first data source name in the list or the next one in the list. <i>Direction</i> can take on only the following values: <ul style="list-style-type: none"> <li>• SQL_FETCH_FIRST</li> <li>• SQL_FETCH_NEXT</li> </ul>
SQLSMALLINT *	<i>NameLength1Ptr</i>	Output	Pointer to location where the maximum number of bytes available to return in the <i>ServerName</i> is stored.
SQLSMALLINT *	<i>NameLength2Ptr</i>	Output	Pointer to location where this function returns the actual number of bytes available to return for the description of the data source.
SQLSMALLINT	<i>BufferLength1</i>	Input	Maximum length of the buffer pointed to by <i>ServerName</i> . This should be less than or equal to SQL_MAX_DSN_LENGTH + 1.
SQLSMALLINT	<i>BufferLength2</i>	Input	Maximum length of the <i>Description</i> buffer.

## Usage

The application can call this function any time by setting *Direction* to either SQL\_FETCH\_FIRST or SQL\_FETCH\_NEXT.

If SQL\_FETCH\_FIRST is specified, the first database in the list is always returned.

If SQL\_FETCH\_NEXT is specified:

- Directly following the 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.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND



## Error conditions

Table 40. *SQLDataSources SQLSTATES*

SQLSTATE	Description	Explanation
01004	Data truncated	The data source name returned in the argument <i>ServerName</i> is longer than the value specified in the argument <i>BufferLength1</i> . The argument <i>NameLength1Ptr</i> contains the length of the full data source name. (Function returns SQL_SUCCESS_WITH_INFO.)  The data source name returned in the argument <i>Description</i> is longer than the value specified in the argument <i>BufferLength2</i> . The argument <i>NameLength2Ptr</i> contains the length of the full data source description. (Function returns SQL_SUCCESS_WITH_INFO.)
58004	Unexpected system failure	Unrecoverable system error.
HY000	General error	An error occurred for which there is no specific SQLSTATE and for which no specific SQLSTATE is defined. The error message returned by <code>SQLError()</code> in the argument <i>ErrorMsg</i> describes the error and its cause.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>ServerName</i> , <i>NameLength1Ptr</i> , <i>Description</i> , or <i>NameLength2Ptr</i> is a null pointer.  Value for the direction that is not valid.
HY013	Unexpected memory handling error	DB2 UDB CLI is unable to access memory required to support the processing or completion of the function.
HY103	Direction option out of range	The value specified for the argument <i>Direction</i> is not equal to SQL_FETCH_FIRST or SQL_FETCH_NEXT.

## Authorization

None.

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/* From CLI sample datasour.c */
/* ... */

#include <stdio.h>
#include <stdlib.h>
#include <sqlcli1.h>
#include "samputil.h"          /* Header file for CLI sample code */

/* ... */

/*****
** main
** - initialize
** - terminate
*****/
int main() {

    SQLHANDLE henv ;
    SQLRETURN rc ;
    SQLCHAR source[SQL_MAX_DSN_LENGTH + 1], description[255] ;

```

```

    SQLSMALLINT buff1, des1 ;
/* ... */

/* allocate an environment handle */
rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;

/* list the available data sources (servers) */
printf( "The following data sources are available:\n" ) ;
printf( "ALIAS NAME          Comment(Description)\n" ) ;
printf( "-----\n" ) ;

while ( ( rc = SQLDataSources( henv,
                               SQL_FETCH_NEXT,
                               source,
                               SQL_MAX_DSN_LENGTH + 1,
                               &buff1,
                               description,
                               255,
                               &des1
                             )
        ) != SQL_NO_DATA_FOUND
      ) printf( "%-30s  %s\n", source, description ) ;

rc = SQLFreeHandle( SQL_HANDLE_ENV, henv ) ;
if ( rc != SQL_SUCCESS ) return( terminate( henv, rc ) ) ;

return( SQL_SUCCESS ) ;

}

```

## References

None.

### Related concepts

“Restrictions for running DB2 UDB CLI in server mode” on page 247

## SQLDescribeCol - Describe column attributes

### Purpose

SQLDescribeCol() returns the result descriptor information (column name, type, precision) for the indicated column in the result set generated by a SELECT statement.

If the application only needs one attribute of the descriptor information, the SQLColAttributes() function can be used in place of SQLDescribeCol().

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

This function (or SQLColAttributes()) is typically called before SQLBindCol().

### Syntax

```

SQLRETURN SQLDescribeCol (SQLHSTMT      hstmt,
                          SQLSMALLINT  icol,
                          SQLCHAR      *szColName,
                          SQLSMALLINT  cbColNameMax,
                          SQLSMALLINT  *pcbColName,
                          SQLSMALLINT  *pfSqlType,

```

```

SQLINTEGER    *pcbColDef,
SQLSMALLINT   *pibScale,
SQLSMALLINT   *pfNullable);

```

## Function arguments

Table 41. *SQLDescribeCol* arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szColName</i>	Output	Pointer to column name buffer.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER *	<i>pcbColDef</i>	Output	Precision of column as defined in the database.  If <i>fSqlType</i> denotes a graphic SQL data type, then this variable indicates the maximum number of double-byte <i>characters</i> the column can hold.
SQLSMALLINT *	<i>pcbColName</i>	Output	Bytes available to return for <i>szColName</i> argument. Truncation of column name ( <i>szColName</i> ) to <i>cbColNameMax</i> - 1 bytes occurs if <i>pcbColName</i> is greater than or equal to <i>cbColNameMax</i> .
SQLSMALLINT *	<i>pfNullable</i>	Output	Indicates whether NULLS are allowed for this column <ul style="list-style-type: none"> <li>• SQL_NO_NULLS.</li> <li>• SQL_NULLABLE.</li> </ul>
SQLSMALLINT *	<i>pfSqlType</i>	Output	SQL data type of column.
SQLSMALLINT *	<i>pibScale</i>	Output	Scale of column as defined in the database (only applies to SQL_DECIMAL, SQL_NUMERIC, SQL_TIMESTAMP).
SQLSMALLINT	<i>cbColNameMax</i>	Input	Size of <i>szColName</i> buffer.
SQLSMALLINT	<i>icol</i>	Input	Column number to be described.

## Usage

Columns are identified by a number and are numbered sequentially from left to right starting with 1, and can be described in any order.

A valid pointer and buffer space must be made available for the *szColName* argument. If a null pointer is specified for any of the remaining pointer arguments, DB2 UDB CLI assumes that the information is not needed by the application and nothing is returned.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- 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 *SQLError()* function.

Table 42. SQLDescribeCol SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated	The column name returned in the argument <i>szColName</i> is longer than the value specified in the argument <i>cbColNameMax</i> . The argument <i>pcbColName</i> contains the length of the full column name. (Function returns SQL_SUCCESS_WITH_INFO.)
07005 *	Not a SELECT statement	The statement associated with the <i>hstmt</i> 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	Column number that is not valid	The value specified for the argument <i>icol</i> is less than 1. The value specified for the argument <i>icol</i> is greater than the number of columns in the result set.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The length specified in argument <i>cbColNameMax</i> is less than 1. The argument <i>szColName</i> or <i>pcbColName</i> is a null pointer.
HY010	Function sequence error	The function is called before calling SQLPrepare() or SQLExecDirect() for the <i>hstmt</i> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HYC00	Driver not capable	The SQL data type of column <i>icol</i> is not recognized by DB2 UDB CLI.

## Example

### References

- “SQLColAttributes - Obtain column attributes” on page 52
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLNumResultCols - Get number of result columns” on page 160
- “SQLPrepare - Prepare a statement” on page 164

#### Related reference

“SQLColAttributes - Obtain column attributes” on page 52

## SQLDescribeParam - Return description of a parameter marker

### Purpose

SQLDescribeParam() returns the description of a parameter marker associated with a prepared SQL statement. This information is also available in the fields of the implementation parameter descriptor (IPD).

## Syntax

```
SQLRETURN SQLDescribeParam (SQLHSTMT          StatementHandle,
                            SQLSMALLINT       ParameterNumber,
                            SQLSMALLINT       *DataTypePtr,
                            SQLINTEGER        *ParameterSizePtr,
                            SQLSMALLINT       *DecimalDigitsPtr,
                            SQLSMALLINT       *NullablePtr);
```

## Function arguments

Table 43. *SQLDescribeParam* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLINTEGER *	<i>ParameterSizePtr</i>	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 *	<i>DataTypePtr</i>	Output	Pointer to a buffer in which to return the SQL data type of the parameter.
SQLSMALLINT *	<i>DecimalDigitsPtr</i>	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 *	<i>NullablePtr</i>	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. <ul style="list-style-type: none"><li>• SQL_NO_NULLS – The parameter does not allow NULL values (this is the default value).</li><li>• SQL_NULLABLE – The parameter allows NULL values.</li><li>• SQL_NULLABLE_UNKNOWN – Cannot determine if the parameter allows NULL values.</li></ul>
SQLSMALLINT	<i>ParameterNumber</i>	Input	Parameter marker number ordered sequentially in increasing parameter order, starting at 1.

## Usage

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

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

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Error conditions

Table 44. *SQLDescribeParam SQLSTATES*

SQLSTATE	Description	Explanation
01000	Warning	Informational message. (Function returns SQL_SUCCESS_WITH_INFO.)
07009	Descriptor index that is not valid	The value specified for the argument <i>ParameterNumber</i> less than 1.  The value specified for the argument <i>ParameterNumber</i> is greater than the number of parameters in the associated SQL statement.  The parameter marker is part of a non-DML statement.  The parameter marker is part of a SELECT list.
08S01	Communication link failure	The communication link between DB2 UDB CLI and the data source to which it is connected fails before the function completes processing.
21S01	Insert value list does not match column list	The number of parameters in the INSERT statement does not match the number of columns in the table named in the statement.
HY000	General error	
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY008	Operation cancelled.	
HY009	Argument value that is not valid	The argument <i>DataTypePtr</i> , <i>ParameterSizePtr</i> , <i>DecimalDigitsPtr</i> , or <i>NullablePtr</i> is a null pointer.
HY010	Function sequence error	The function is called before calling SQLPrepare() or SQLExecDirect() for the <i>StatementHandle</i> .
HY013	Unexpected memory handling error	The function call cannot be processed because the underlying memory objects can not be accessed, possibly because of low memory conditions.

## Restrictions

None.

## References

- “SQLBindParam - Bind a buffer to a parameter marker” on page 38
- “SQLCancel - Cancel statement” on page 50
- “SQLExecute - Execute a statement” on page 83
- “SQLPrepare - Prepare a statement” on page 164

## SQLDisconnect - Disconnect from a data source

### Purpose

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

After calling this function, either call SQLConnect() to connect to another database, or call SQLFreeConnect().

### Syntax

```
SQLRETURN SQLDisconnect (SQLHDBC hdbc);
```

### Function arguments

Table 45. SQLDisconnect arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Connection handle

### Usage

If an application calls SQLDisconnect before it has freed all the statement handles associated with the connection, DB2 UDB 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 is encountered on the clean up after 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 *hdbc* to make another SQLConnect() request.

If the *hdbc* is participating in a DUOW two-phase commit connection, the disconnect might not occur immediately. The actual disconnect occurs at the next commit issued for the distributed transaction.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 46. 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 not open	The connection specified in the argument <i>hdbc</i> is not open.

Table 46. *SQLDisconnect SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
25000	Transaction state that is not valid	There is a transaction in process on the connection specified by the argument <i>hdbc</i> . The transaction remains active, and the connection cannot be disconnected.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## Example

Refer to the example in “SQLAllocEnv - Allocate environment handle” on page 24.

## References

- “SQLAllocConnect - Allocate connection handle” on page 21
- “SQLConnect - Connect to a data source” on page 62
- “SQLTransact - Commit or roll back transaction” on page 217

## SQLDriverConnect - (Expanded) Connect to a data source

### Purpose

SQLDriverConnect() is an alternative to SQLConnect(). Both functions establish a connection to the target database, but SQLDriverConnect() uses a connection string to determine the data source name, user ID and password. The functions are the same; both are supported for compatibility purposes.

### Syntax

```
SQLRETURN SQLDriverConnect (SQLHDBC          ConnectionHandle,
                             SQLHWND         WindowHandle,
                             SQLCHAR         *InConnectionString,
                             SQLSMALLINT     StringLength1,
                             SQLCHAR         *OutConnectionString,
                             SQLSMALLINT     BufferLength,
                             SQLSMALLINT     *StringLength2Ptr,
                             SQLSMALLINT     DriverCompletion);
```

### Function arguments

Table 47. *SQLDriverConnect arguments*

Data type	Argument	Use	Description
SQLCHAR *	<i>InConnectionString</i>	Input	A full, partial, or empty (null pointer) connection string.
SQLCHAR *	<i>OutConnectionString</i>	Output	Pointer to buffer for the completed connection string. If the connection is established successfully, this buffer contains the completed connection string.
SQLHDBC	<i>ConnectionHandle</i>	Input	Connection handle.
SQLHWND	<i>hwindow</i>	Input	For DB2 Universal Database™ for Linux®, UNIX®, and Windows, this is the parent handle. On i5/OS, it is ignored.



Table 47. *SQLDriverConnect* arguments (continued)

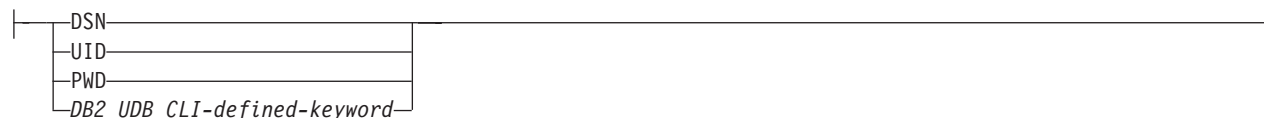
Data type	Argument	Use	Description
SQLSMALLINT *	<i>StringLength2Ptr</i>	Output	Pointer to the number of bytes available to return in the <i>OutConnectionString</i> buffer.  If the value of <i>StringLength2Ptr</i> is greater than or equal to <i>BufferLength</i> , the completed connection string in <i>OutConnectionString</i> is truncated to <i>BufferLength</i> - 1 bytes.
SQLSMALLINT	<i>DriverCompletion</i>	Input	Indicates when DB2 UDB CLI should prompt the user for more information.  Possible values: <ul style="list-style-type: none"> <li>• SQL_DRIVER_COMPLETE</li> <li>• SQL_DRIVER_COMPLETE_REQUIRED</li> <li>• SQL_DRIVER_NOPROMPT</li> </ul>
SQLSMALLINT	<i>BufferLength</i>	Input	Maximum size of the buffer pointed to by <i>OutConnectionString</i> .
SQLSMALLINT	<i>StringLength1</i>	Input	Length of <i>InConnectionString</i> .

## Usage

The connection string is used to pass one or more values that are needed to complete a connection. The contents of the connection string and the value of *DriverCompletion* determine how the connection should be established.



## Connection string syntax



Each keyword above has an attribute that is equal to:

**DSN** Data source name. The name or alias-name of the database. The data source name is required if *DriverCompletion* is equal to SQL\_DRIVER\_NOPROMPT.

**UID** Authorization-name (user identifier).

**PWD** The password that corresponds to the authorization name. If there is no password for the user ID, empty is specified (PWD=;).

iSeries currently has no DB2 UDB CLI-defined keywords.

The value of *DriverCompletion* is verified to be valid, but all result in the same behavior. A connection is attempted with the information that is contained in the connection string. If there is not enough information, SQL\_ERROR is returned.

As soon as 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`

## Error conditions

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

Table 48. `SQLDriverConnect` `SQLSTATES`

SQLSTATE	Description	Explanation
01004	Data truncated	The buffer <code>szConnstrOut</code> is not large enough to hold the entire connection string. The argument <code>StringLength2Ptr</code> contains the actual length of the connection string available for return. (Function returns <code>SQL_SUCCESS_WITH_INFO</code> )
01S00	Connection string attribute that is not valid	A keyword or attribute value that is not valid is specified in the input connection string, but the connection to the data source is successful anyway because one of the following situations occurs: <ul style="list-style-type: none"> <li>• The unrecognized keyword is ignored.</li> <li>• The attribute value that is not valid is ignored, the default value is used instead.</li> </ul> (Function returns <code>SQL_SUCCESS_WITH_INFO</code> )
HY009	Argument value that is not valid	The argument <code>InConnectionString</code> , <code>OutConnectionString</code> , or <code>StringLength2PTR</code> is a null pointer.  The argument <code>DriverCompletion</code> is not equal to 1.
HY090	String or buffer length that is not valid	The value specified for <code>StringLength1</code> is less than 0, but not equal to <code>SQL_NTS</code> .  The value specified for <code>BufferLength</code> is less than 0.
HY110	Driver completion that is not valid	The value specified for the argument <code>fCompletion</code> is not equal to one of the valid values.

## Restrictions

None.

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```
/* From CLI sample drivrcon.c */
/* ... */
/*****
**   drv_connect - Prompt for connect options and connect           **
```

```

*****/

int
drv_connect(SQLHENV henv,
            SQLHDBC * hdbc,
            SQLCHAR con_type)
{
    SQLRETURN      rc;
    SQLCHAR        server[SQL_MAX_DSN_LENGTH + 1];
    SQLCHAR        uid[MAX_UID_LENGTH + 1];
    SQLCHAR        pwd[MAX_PWD_LENGTH + 1];
    SQLCHAR        con_str[255];
    SQLCHAR        buffer[255];
    SQLSMALLINT    outlen;

    printf("Enter Server Name:\n");
    gets((char *) server);
    printf("Enter User Name:\n");
    gets((char *) uid);
    printf("Enter Password Name:\n");
    gets((char *) pwd);

    /* Allocate a connection handle */
    SQLAllocHandle( SQL_HANDLE_DBC,
                   henv,
                   hdbc
                   );
    CHECK_HANDLE( SQL_HANDLE_DBC, *hdbc, rc);

    sprintf((char *)con_str, "DSN=%s;UID=%s;PWD=%s;",
           server, uid, pwd);

    rc = SQLDriverConnect(*hdbc,
                          (SQLHWND) NULL,
                          con_str,
                          SQL_NTS,
                          buffer, 255, &outlen,
                          SQL_DRIVER_NOPROMPT);
    if (rc != SQL_SUCCESS) {
        printf("Error while connecting to database, RC= %ld\n", rc);
        CHECK_HANDLE( SQL_NULL_HENV, *hdbc, rc);
        return (SQL_ERROR);
    } else {
        printf("Successful Connect\n");
        return (SQL_SUCCESS);
    }
}

```

### Related reference

“SQLConnect - Connect to a data source” on page 62

## SQLEndTran - Commit or roll back a transaction

### Purpose

SQLEndTran() commits or rolls back the current transaction in the connection.

All changes to the database performed on the connection since connect time or the previous call to SQLEndTran() (whichever is the most recent) are committed or rolled back.

If a transaction is active on a connection, the application must call SQLEndTran() before it can disconnect from the database.

## Syntax

```
SQLRETURN SQLEndTran (SQLSMALLINT hType,  
                     SQLINTEGER handle,  
                     SQLSMALLINT fType);
```

## Function arguments

Table 49. *SQLEndTran* arguments

Data type	Argument	Use	Description
SQLINTEGER	<i>handle</i>	Input	Handle to use when performing the COMMIT or ROLLBACK.
SQLSMALLINT	<i>fType</i>	Input	Wanted action for the transaction. The value for this argument must be one of: <ul style="list-style-type: none"><li>• SQL_COMMIT</li><li>• SQL_ROLLBACK</li><li>• SQL_COMMIT_HOLD</li><li>• SQL_ROLLBACK_HOLD</li><li>• SQL_SAVEPOINT_NAME_ROLLBACK</li><li>• SQL_SAVEPOINT_NAME_RELEASE</li></ul>
SQLSMALLINT	<i>hType</i>	Input	Type of handle It must contain SQL_HANDLE_ENV or SQL_HANDLE_DBC.

## Usage

Completing a transaction with SQL\_COMMIT or SQL\_ROLLBACK has the following effects:

- Statement handles are still valid after a call to SQLEndTran().
- Cursor names, bound parameters, and column bindings survive transactions.
- Open cursors are closed, and any result sets that are pending retrieval are discarded.

Completing the transaction with SQL\_COMMIT\_HOLD or SQL\_ROLLBACK\_HOLD still commits or rolls back the database changes, but does not cause cursors to be closed.

If no transaction is currently active on the connection, calling SQLEndTran() has no effect on the database server and returns SQL\_SUCCESS.

SQLEndTran() might fail while executing the COMMIT or ROLLBACK due to a loss of connection. In this case the application might be unable to determine whether the COMMIT or ROLLBACK has been processed, and a database administrator's help might be required. Refer to the Database Management System (DBMS) product information for more information about transaction logs and other transaction management tasks.

When using either SQL\_SAVEPOINT\_NAME\_ROLLBACK or SQL\_SAVEPOINT\_NAME\_RELEASE, you must already have set the savepoint name using SQLSetConnectAttr.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 50. *SQLEndTran SQLSTATES*

SQLSTATE	Description	Explanation
08003	Connection not open	The <i>hdbc</i> is not in a connected state.
08007	Connection failure during transaction	The connection associated with the <i>hdbc</i> fails during the processing of the function during the processing of the function and it cannot be determined whether the requested COMMIT or ROLLBACK occurs before the failure.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	SQL_SAVEPOINT_NAME_ROLLBACK or SQL_SAVEPOINT_NAME_RELEASE is used, but the savepoint name is not established by calling SQLSetConnectAttr() for attribute SQL_ATTR_SAVEPOINT_NAME.
HY012	Transaction operation state that is not valid	The value specified for the argument <i>fType</i> is neither SQL_COMMIT nor SQL_ROLLBACK.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## SQLError - Retrieve error information

### Purpose

SQLError() returns the diagnostic information associated with the most recently called DB2 UDB CLI function for a particular statement, connection or environment handle.

The information consists of a standardized SQLSTATE, an error code, and a text message. Refer to "Diagnostics in a DB2 UDB CLI application" on page 14 for more information.

Call SQLError() after receiving a return code of SQL\_ERROR or SQL\_SUCCESS\_WITH\_INFO from another function call.

### Syntax

```
SQLRETURN SQLError (SQLHENV      henv,  
                   SQLHDBC      hdbc,  
                   SQLHSTMT     hstmt,  
                   SQLCHAR      *szSqlState,  
                   SQLINTEGER    *pfNativeError,  
                   SQLCHAR      *szErrorMsg,  
                   SQLSMALLINT   cbErrorMsgMax,  
                   SQLSMALLINT   *pcbErrorMsg);
```

## Function arguments

Table 51. *SQLError arguments*

Data type	Argument	Use	Description
SQLCHAR *	<i>szErrorMsg</i>	Output	Pointer to buffer to contain the implementation defined message text. In DB2 UDB CLI, only the DBMS generated messages is returned; DB2 UDB CLI itself does not return any message text describing the problem.
SQLCHAR *	<i>szSqlState</i>	Output	SQLSTATE as a string of 5 characters terminated by a null character. The first 2 characters indicate error class; the next 3 indicate subclass. The values correspond directly to SQLSTATE values defined in the X/Open SQL CAE specification and the ODBC specification, augmented with IBM specific and product specific SQLSTATE values.
SQLHDBC	<i>hdbc</i>	Input	Database connection handle. To obtain diagnostic information associated with a connection, pass a valid database connection handle, and set <i>hstmt</i> to SQL_NULL_HSTMT. The <i>henv</i> argument is ignored.
SQLHENV	<i>henv</i>	Input	Environment handle. To obtain diagnostic information associated with an environment, pass a valid environment handle. Set <i>hdbc</i> to SQL_NULL_HDBC. Set <i>hstmt</i> to SQL_NULL_HSTMT.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle. To obtain diagnostic information associated with a statement, pass a valid statement handle. The <i>henv</i> and <i>hdbc</i> arguments are ignored.
SQLINTEGER *	<i>pfNativeError</i>	Output	Native error code. In DB2 UDB CLI, the <i>pfNativeError</i> argument contains the SQLCODE value returned by the Database Management System (DBMS). If the error is generated by DB2 UDB CLI and not the DBMS, this field is set to -99999.
SQLSMALLINT *	<i>pcbErrorMsg</i>	Output	Pointer to total number of bytes available to return to the <i>szErrorMsg</i> buffer.
SQLSMALLINT	<i>cbErrorMsgMax</i>	Input	Maximum (that is, the allocated) length of the buffer <i>szErrorMsg</i> . The recommended length to allocate is SQL_MAX_MESSAGE_LENGTH + 1.

## Usage

The SQLSTATEs are those defined by the X/OPEN SQL CAE and the X/Open SQL CLI snapshot, augmented with IBM specific and product specific SQLSTATE values.

:

- To obtain diagnostic information associated with an environment, pass a valid environment handle. Set *hdbc* to SQL\_NULL\_HDBC. Set *hstmt* to SQL\_NULL\_HSTMT.

- To obtain diagnostic information associated with a connection, pass a valid database connection handle, and set *hstmt* to SQL\_NULL\_HSTMT. The *henv* argument is ignored.
- To obtain diagnostic information associated with a statement, pass a valid statement handle. The *henv* and *hdbc* arguments are ignored.

If diagnostic information generated by one DB2 UDB CLI function is not retrieved before a function other than `SQLError()` is called with the same handle, the information for the previous function call is lost. This is true whether diagnostic information is generated for the second DB2 UDB CLI function call.

To avoid truncation of the error message, declare a buffer length of `SQL_MAX_MESSAGE_LENGTH + 1`. The message text is never longer than this.

## Return codes

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND
- SQL\_SUCCESS

## Diagnostics

SQLSTATEs are not defined because `SQLError()` does not generate diagnostic information for itself. `SQL_ERROR` is returned if argument `szSqlState`, `pfNativeError`, `szErrorMsg`, or `pcbErrorMsg` is a null pointer.

## Example

### Related concepts

“Diagnostics in a DB2 UDB CLI application” on page 14

This topic deals with warning or error conditions generated within an application.

## SQLExecDirect - Execute a statement directly

### Purpose

`SQLExecDirect()` directly executes the specified SQL statement. The statement can only be processed once. Also, the connected database server must be able to prepare the statement.

### Syntax

```
SQLRETURN SQLExecDirect (SQLHSTMT      hstmt,
                        SQLCHAR        *szSqlStr,
                        SQLINTEGER      cbSqlStr);
```

### Function arguments

Table 52. *SQLExecDirect* arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szSqlStr</i>	Input	SQL statement string. The connected database server must be able to prepare the statement.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle. There must not be an open cursor associated with <i>hstmt</i> . See “SQLFreeStmt - Free (or reset) a statement handle” on page 102 for more information.

Table 52. *SQLExecDirect* arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>cbSqlStr</i>	Input	Length of contents of <i>szSqlStr</i> argument. The length must be set to either the exact length of the statement, or if the statement is null-terminated, set to SQL_NTS.

## Usage

The SQL statement cannot be a COMMIT or ROLLBACK. Instead, `SQLTransact()` must be called to issue COMMIT or ROLLBACK. For more information about supported SQL statements refer to Table 1 on page 3.

The SQL statement string might contain parameter markers. A parameter marker is represented by a "?" character, and indicates a position in the statement where the value of an application variable is to be substituted, when `SQLExecDirect()` is called. `SQLBindParam()` binds (or associates) an application variable to each parameter marker, to indicate if any data conversion should be performed at the time the data is transferred. All parameters must be bound before calling `SQLExecDirect()`.

If the SQL statement is a SELECT, `SQLExecDirect()` generates a cursor name, and open the cursor. If the application has used `SQLSetCursorName()` to associate a cursor name with the statement handle, DB2 UDB CLI associates the application generated cursor name with the internally generated one.

To retrieve a row from the result set generated by a SELECT statement, call `SQLFetch()` after `SQLExecDirect()` returns successfully.

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

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

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

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 53. *SQLExecDirect* SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value	The argument <i>szSqlStr</i> is a null pointer.  The argument <i>cbSqlStr</i> is less than 1, but not equal to SQL_NTS.



Table 53. *SQLExecDirect SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
HY010	Function sequence error	Either no connection or there is an open cursor for this statement handle.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

**Note:** There are many other SQLSTATE values that can be generated by the Database Management System (DBMS), on processing of the statement.

## Example

Refer to the `SQLFetch()` "" on page 88.

## References

- "SQLExecute - Execute a statement"
- "SQLFetch - Fetch next row" on page 87
- "SQLSetParam - Set parameter" on page 200

## SQLExecute - Execute a statement

### Purpose

`SQLExecute()` executes a statement, that is successfully prepared using `SQLPrepare()`, once or multiple times. The statement is processed using the current values of any application variables that were bound to parameter markers by `SQLBindParam()`.

### Syntax

```
SQLRETURN SQLExecute (SQLHSTMT      hstmt);
```

### Function arguments

Table 54. *SQLExecute arguments*

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle. There must not be an open cursor associated with <i>hstmt</i> , see "SQLFreeStmt - Free (or reset) a statement handle" on page 102 for more information.

### Usage

The SQL statement string might contain parameter markers. A parameter marker is represented by a "?" character, and indicates a position in the statement where the value of an application variable is to be substituted, when `SQLExecute()` is called. `SQLBindParam()` is used to bind (or associate) an application variable to each parameter marker, and to indicate if any data conversion should be performed at the time the data is transferred. All parameters must be bound before calling `SQLExecute()`.

As soon as the application has processed the results from the `SQLExecute()` call, it can process the statement again with new (or the same) values in the application variables.

A statement processed by `SQLExecDirect()` cannot be reprocessed by calling `SQLExecute()`; `SQLPrepare()` must be called first.

If the prepared SQL statement is a `SELECT`, `SQLExecute()` generates a cursor name, and opens the cursor. If the application has used `SQLSetCursorName()` to associate a cursor name with the statement handle, DB2 UDB CLI associates the application generated cursor name with the internally generated cursor name.

To process a `SELECT` statement more than once, the application must close the cursor by calling call `SQLFreeStmt()` with the `SQL_CLOSE` option. There must not be an open cursor on the statement handle when calling `SQLExecute()`.

To retrieve a row from the result set generated by a `SELECT` statement, call `SQLFetch()` after `SQLExecute()` returns successfully.

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.

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

`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()` (see Table 53 on page 82) except for `HY009`, and with the addition of the `SQLSTATES` in the following table.

Table 55. `SQLExecute SQLSTATES`

SQLSTATE	Description	Explanation
HY010	Function sequence error	The specified <i>hstmt</i> is not in prepared state. <code>SQLExecute()</code> is called without first calling <code>SQLPrepare</code> .
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

**Note:** There are many other `SQLSTATE` values that can be generated by the Database Management System (DBMS), on processing of the statement.

## Example

Refer to the `SQLPrepare()` "" on page 166

## References

- “`SQLExecDirect` - Execute a statement directly” on page 81
- “`SQLBindCol` - Bind a column to an application variable” on page 29
- “`SQLPrepare` - Prepare a statement” on page 164
- “`SQLFetch` - Fetch next row” on page 87

- “SQLSetParam - Set parameter” on page 200

## SQLExtendedFetch - Fetch array of rows

### Purpose

SQLExtendedFetch() extends the function of SQLFetch() by returning a block of data containing multiple rows (called a *rowset*), in the form of an array, for each bound column. The size of the rowset is determined by the SQL\_ROWSET\_SIZE attribute on an SQLSetStmtAttr() call.

To fetch one row of data at a time, an application should call SQLFetch().

### Syntax

```
SQLRETURN SQLExtendedFetch (SQLHSTMT StatementHandle,
                             SQLSMALLINT FetchOrientation,
                             SQLINTEGER FetchOffset,
                             SQLINTEGER *RowCountPtr,
                             SQLSMALLINT *RowStatusArray);
```

### Function arguments

Table 56. SQLExtendedFetch arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLINTEGER *	<i>RowCountPtr</i>	Output	Number of the rows actually fetched. If an error occurs during processing, <i>RowCountPtr</i> points to the ordinal position of the row (in the rowset) that precedes the row where the error occurred. If an error occurs retrieving the first row <i>RowCountPtr</i> points to the value 0.
SQLINTEGER	<i>FetchOffset</i>	Input	Row offset for relative positioning.
SQLSMALLINT *	<i>RowStatusArray</i>	Output	An array of status values. The number of elements must equal the number of rows in the rowset (as defined by the SQL_ROWSET_SIZE attribute). A status value for each row fetched is returned: <ul style="list-style-type: none"> <li>• SQL_ROW_SUCCESS</li> </ul> <p>If the number of rows fetched is less than the number of elements in the status array (that is, less than the rowset size), the remaining status elements are set to SQL_ROW_NOROW.</p> <p>DB2 UDB CLI cannot detect whether a row has been updated or deleted since the start of the fetch. Therefore, the following ODBC defined status values are not reported:</p> <ul style="list-style-type: none"> <li>• SQL_ROW_DELETED</li> <li>• SQL_ROW_UPDATED</li> </ul>
SQLSMALLINT	<i>FetchOrientation</i>	Input	Fetch orientation. See Table 61 on page 93 for possible values.

### Usage

SQLExtendedFetch() is used to perform an array fetch of a set of rows. An application specifies the size of the array by calling SQLSetStmtAttr() with the SQL\_ROWSET\_SIZE attribute.

Before `SQLExtendedFetch()` is called the first time, the cursor is positioned before the first row. After `SQLExtendedFetch()` is called, the cursor is positioned on the row in the result set corresponding to the last row element in the rowset just retrieved.

For any columns in the result set that have been bound by the `SQLBindCol()` function, DB2 UDB CLI converts the data for the bound columns as necessary and stores it in the locations bound to these columns. The result set must be bound in a row-wise fashion. This means that the values for all the columns of the first row are contiguous, followed by the values of the second row, and so on. Also, if indicator variables are used, they are all returned in one contiguous storage location.

When using this procedure to retrieve multiple rows, all columns must be bound, and the storage must be contiguous. When using this function to retrieve rows from an SQL procedure result set, only the `SQL_FETCH_NEXT` orientation is supported. The user is responsible for allocating enough storage for the number of rows that are specified in `SQL_ROWSET_SIZE`.

The cursor must be a scrollable cursor for `SQLExtendedFetch()` to use any orientation other than `SQL_FETCH_NEXT`. See “`SQLSetStmtAttr - Set a statement attribute`” on page 201 for information about setting the `SQL_ATTR_CURSOR_SCROLLABLE` attribute.

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

## Error conditions

Table 57. `SQLExtendedFetch` `SQLSTATEs`

<code>SQLSTATE</code>	Description	Explanation
<code>HY009</code>	Argument value that is not valid	<p>The argument value <code>RowCountPtr</code> or <code>RowStatusArray</code> is a null pointer.</p> <p>The value specified for the argument <code>FetchOrientation</code> is not recognized.</p>
<code>HY010</code>	Function sequence error	<p><code>SQLExtendedFetch()</code> is called for an <code>StatementHandle</code> after <code>SQLFetch()</code> is called and before <code>SQLFreeStmt()</code> has been called with the <code>SQL_CLOSE</code> option.</p> <p>The function is called before calling <code>SQLPrepare()</code> or <code>SQLExecuteDirect()</code> for the <code>StatementHandle</code>.</p> <p>The function is called while in a data-at-processing (<code>SQLParamData()</code>, <code>SQLPutData()</code>) operation.</p>

## Restrictions

None.

## References

- “`SQLBindCol - Bind a column to an application variable`” on page 29
- “`SQLExecute - Execute a statement`” on page 83
- “`SQLExecuteDirect - Execute a statement directly`” on page 81

- “SQLFetch - Fetch next row”

## SQLFetch - Fetch next row

### Purpose

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

SQLFetch() can be used to receive the data directly into variables you specify with SQLBindCol(), or the columns can be received individually after the fetch, by calling SQLGetData(). Data conversion is also performed when SQLFetch() is called, if conversion is indicated when the column is bound.

### Syntax

```
SQLRETURN SQLFetch (SQLHSTMT hstmt);
```

### Function arguments

Table 58. SQLFetch arguments

Data type	argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle

### Usage

SQLFetch() can only be called if the most recently processed statement on *hstmt* is a SELECT.

The number of application variables bound with SQLBindCol() must not exceed the number of columns in the result set; otherwise SQLFetch() fails.

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() can then be called to obtain all of the columns individually. Data in unbound columns is discarded when SQLFetch() advances the cursor to the next row.

If any bound variables are not large enough to hold the data returned by SQLFetch(), the data is truncated. If character data is truncated, and the *SQLSetEnvAttr()* attribute *SQL\_ATTR\_TRUNCATION\_RTNC* is set to *SQL\_TRUE*, then the CLI return code *SQL\_SUCCESS\_WITH\_INFO* is returned, along with an *SQLSTATE* that indicates truncation. Note that the default is *SQL\_FALSE* for *SQL\_ATTR\_TRUNCATION\_RTNC*. Also, in the case of character data truncation, the *SQLBindCol()* deferred output argument *pcbValue* contains the actual length of the column data retrieved from the server. The application should compare the output length to the input length (*pcbValue* and *cbValueMax* arguments from *SQLBindCol()*) to determine which character columns have been truncated.

Truncation of numeric data types is not reported 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 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 data transferred between DB2 UDB CLI and the application is never null-terminated.

When all the rows have been retrieved from the result set, or the remaining rows are not needed, `SQLFreeStmt()` should be called to close the cursor and discard the remaining data and associated resources.

### Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `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.

### Diagnostics

Table 59. *SQLFetch SQLSTATES*

SQLSTATE	Description	Explanation
01004	Data truncated	The data returned for one or more columns is truncated. String values are right truncated. (SQL_SUCCESS_WITH_INFO is returned if no error occurred.)
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	The specified <i>hstmt</i> is not in an processed state. The function is called without first calling <code>SQLExecute</code> or <code>SQLExecDirect</code> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

### Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/*****
** file = fetch.c
**
** Example of executing an SQL statement.
** SQLBindCol & SQLFetch is used to retrieve data from the result set
** directly into application storage.
**
** Functions used:
**
**      SQLAllocConnect      SQLFreeConnect
**      SQLAllocEnv         SQLFreeEnv
**      SQLAllocStmt        SQLFreeStmt
**      SQLConnect          SQLDisconnect
**
**      SQLBindCol          SQLFetch
**      SQLTransact         SQLExecDirect
**      SQLError
**
*****/

#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

```

```

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
              SQLHDBC *hdbc);

int terminate(SQLHENV henv,
             SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt);

int check_error (SQLHENV    henv,
                SQLHDBC    hdbc,
                SQLHSTMT   hstmt,
                SQLRETURN   rc);

/*****
** main
** - initialize
** - terminate
*****/
int main()
{
    SQLHENV    henv;
    SQLHDBC    hdbc;
    SQLCHAR    sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN   rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));

    {SQLHSTMT   hstmt;
      SQLCHAR    sqlstmt[]="SELECT deptname, location from org where division = 'Eastern'";
      SQLCHAR    deptname[15],
                location[14];
      SQLINTEGER rlength;

      rc = SQLAllocStmt(hdbc, &hstmt);
      if (rc != SQL_SUCCESS )
          check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

      rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
      if (rc != SQL_SUCCESS )
          check_error (henv, hdbc, hstmt, rc);

      rc = SQLBindCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15,
                    &rlength);
      if (rc != SQL_SUCCESS )
          check_error (henv, hdbc, hstmt, rc);
      rc = SQLBindCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14,
                    &rlength);
      if (rc != SQL_SUCCESS )
          check_error (henv, hdbc, hstmt, rc);

      printf("Departments in Eastern division:\n");
      printf("DEPTNAME      Location\n");
      printf("-----\n");

      while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
      {
          printf("%-14.14s %-13.13s \n", deptname, location);
      }
      if (rc != SQL_NO_DATA_FOUND )
          check_error (henv, hdbc, hstmt, rc);
    }
}

```

```

        rc = SQLFreeStmt(hstmt, SQL_DROP);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
    }

    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (0);
}/* end main */

/*****
** initialize
** - allocate environment handle
** - allocate connection handle
** - prompt for server, user id, & password
** - connect to server
*****/

int initialize(SQLHENV *henv,
              SQLHDBC *hdbc)
{
SQLCHAR      server[SQL_MAX_DSN_LENGTH],
             uid[30],
             pwd[30];
SQLRETURN    rc;

    rc = SQLAllocEnv (henv);          /* allocate an environment handle */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    rc = SQLAllocConnect (*henv, hdbc); /* allocate a connection handle */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

    printf("Enter Server Name:\n");
    gets(server);
    printf("Enter User Name:\n");
    gets(uid);
    printf("Enter Password Name:\n");
    gets(pwd);

    if (uid[0] == '\0')
    {
        rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }
    else
    {
        rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
    }

    return(SQL_SUCCESS);
}/* end initialize */

/*****
** terminate
** - disconnect
** - free connection handle
** - free environment handle
*****/
int terminate(SQLHENV henv,

```



```

        SQLHDBC hdbc)
{
SQLRETURN rc;

    rc = SQLDisconnect (hdbc);          /* disconnect from database */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeConnect (hdbc);        /* free connection handle */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeEnv (henv);           /* free environment handle */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);

    return(rc);
}/* end terminate */

/*****
** - print_error - call SQLError(), display SQLSTATE and message
*****/

int print_error (SQLHENV henv,
                SQLHDBC hdbc,
                SQLHSTMT hstmt)
{
SQLCHAR buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER sqlcode;
SQLSMALLINT length;

    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                    SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR ****\n");
        printf("        SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };

    return ( SQL_ERROR);
} /* end print_error */

/*****
** - check_error - call print_error(), checks severity of return code
*****/

int check_error (SQLHENV henv,
                SQLHDBC hdbc,
                SQLHSTMT hstmt,
                SQLRETURN frc)
{
SQLRETURN rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
    case SQL_SUCCESS : break;
    case SQL_ERROR :
    case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
        else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
        exit(frc);
    }
}

```

```

        break;
    case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
    case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
    default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);
} /* end check_error */

```

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLExecute - Execute a statement” on page 83
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLGetCol - Retrieve one column of a row of the result set” on page 103
- “SQLFetchScroll - Fetch from a scrollable cursor”

## SQLFetchScroll - Fetch from a scrollable cursor

### Purpose

SQLFetchScroll() positions the cursor based on the requested orientation, then retrieves any bound columns.

SQLFetchScroll() can be used to receive the data directly into variables you specify with SQLBindCol(), or the columns can be received individually after the fetch, by calling SQLGetData(). Data conversion is also performed when SQLFetchScroll() is called, if conversion is indicated when the column is bound.

### Syntax

```

SQLRETURN SQLFetchScroll (SQLHSTMT    hstmt,
                          SQLSMALLINT fOrient,
                          SQLINTEGER   fOffset);

```

### Function arguments

Table 60. SQLFetchScroll arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLSMALLINT	<i>fOrient</i>	Input	Fetch orientation. See Table 61 on page 93 for possible values.
SQLINTEGER	<i>fOffset</i>	Input	Row offset for relative positioning.

### Usage

SQLFetchScroll() can only be called if the most recently processed statement on *hstmt* is a SELECT.

SQLFetchScroll() acts like SQLFetch(), except the *fOrient* parameter positions the cursor before any data is retrieved. The cursor must be a scrollable cursor for SQLFetchScroll() to use any orientation other than SQL\_FETCH\_NEXT. See “SQLSetStmtAttr - Set a statement attribute” on page 201 for information about setting the SQL\_ATTR\_CURSOR\_SCROLLABLE attribute.

When using this function to retrieve rows from an SQL procedure result set, only the SQL\_FETCH\_NEXT orientation is supported.

- | SQLFetchScroll() supports array fetch, an alternative to the array fetch support provided by
- | SQLExtendedFetch(). See the SQLExtendedFetch() topic for details on array fetch.

The information returned in the *RowCountPtr* and *RowStatusArray* parameters of SQLExtendedFetch() are handled by SQLFetchScroll() as follows:

- *RowCountPtr*: SQLFetchScroll() returns the number of rows fetched in the buffer pointed to by the SQL\_ATTR\_ROWS\_FETCHED\_PTR statement attribute.
- *RowStatusArray*: SQLFetchScroll() returns the array of statuses for each row in the buffer pointed to by the SQL\_ATTR\_ROW\_STATUS\_PTR statement attribute.

Table 61. Statement attributes

<i>fOrient</i>	Description
SQL_FETCH_FIRST	Move to the first row of the result set.
SQL_FETCH_LAST	Move to the last row of the result set.
SQL_FETCH_NEXT	Move to the row following the current cursor position.
SQL_FETCH_PRIOR	Move to the row preceding the current cursor position.
SQL_FETCH_RELATIVE	If <i>fOffset</i> is: <ul style="list-style-type: none"> <li>• Positive, advance the cursor that number of rows.</li> <li>• Negative, back up the cursor that number of rows.</li> <li>• Zero, do not move the cursor.</li> </ul>

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

## Diagnostics

Table 62. SQLFetchScroll SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated	The data returned for one or more columns is truncated. String values are right truncated. (SQL_SUCCESS_WITH_INFO is returned if no error occurred.)
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	Orientation that is not valid.

Table 62. *SQLFetchScroll SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
HY010	Function sequence error	The specified <i>hstmt</i> is not in an processed state. The function is called without first calling <code>SQLExecute</code> or <code>SQLExecDirect</code> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## References

- “`SQLBindCol` - Bind a column to an application variable” on page 29
- “`SQLExecute` - Execute a statement” on page 83
- “`SQLExecDirect` - Execute a statement directly” on page 81
- “`SQLExtendedFetch` - Fetch array of rows” on page 85
- “`SQLGetCol` - Retrieve one column of a row of the result set” on page 103
- “`SQLFetch` - Fetch next row” on page 87
- “`SQLSetStmtAttr` - Set a statement attribute” on page 201

### Related reference

“`SQLSetStmtAttr` - Set a statement attribute” on page 201

## SQLForeignKeys - Get the list of foreign key columns

### Purpose

`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 that is generated by a query.

### Syntax

```
SQLRETURN SQLForeignKeys (SQLHSTMT StatementHandle,
                          SQLCHAR *PKCatalogName,
                          SQLSMALLINT NameLength1,
                          SQLCHAR *PKSchemaName,
                          SQLSMALLINT NameLength2,
                          SQLCHAR *PKTableName,
                          SQLSMALLINT NameLength3,
                          SQLCHAR *FKCatalogName,
                          SQLSMALLINT NameLength4,
                          SQLCHAR *FKSchemaName,
                          SQLSMALLINT NameLength5,
                          SQLCHAR *FKTableName,
                          SQLSMALLINT NameLength6);
```

### Function arguments

Table 63. *SQLForeignKeys arguments*

Data type	Argument	Use	Description
SQLCHAR *	<i>FKCatalogName</i>	Input	Catalog qualifier of the table containing the foreign key. This must be a NULL pointer or a zero length string.
SQLCHAR *	<i>FKSchemaName</i>	Input	Schema qualifier of the table containing the foreign key.
SQLCHAR *	<i>FKTableName</i>	Input	Name of the table containing the foreign key.

Table 63. *SQLForeignKeys* arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>PKCatalogName</i>	Input	Catalog qualifier of the primary key table. This must be a NULL pointer or a zero length string.
SQLCHAR *	<i>PKSchemaName</i>	Input	Schema qualifier of the primary key table.
SQLCHAR *	<i>PKTableName</i>	Input	Name of the table name containing the primary key.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT	<i>NameLength2</i>	Input	Length of <i>PKSchemaName</i> .
SQLSMALLINT	<i>NameLength3</i>	Input	Length of <i>PKTableName</i> .
SQLSMALLINT	<i>NameLength6</i>	Input	Length of <i>FKTableName</i> .
SQLSMALLINT	<i>NameLength1</i>	Input	Length of <i>PKCatalogName</i> . This must be set to 0.
SQLSMALLINT	<i>NameLength4</i>	Input	Length of <i>FKCatalogName</i> . This must be set to 0.
SQLSMALLINT	<i>NameLength5</i>	Input	Length of <i>FKSchemaName</i> .

## Usage

If *PKTableName* contains a table name, and *FKTableName* is an empty string, *SQLForeignKeys()* returns a result set that contains 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 that contains 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 that is associated with a table name is not specified, then for the schema name the default is the one currently in effect for the current connection.

Table 64 lists the columns of the result set generated by the *SQLForeignKeys()* call. If the foreign keys that are 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 that are associated with a foreign key are requested, the result set is ordered by *PKTABLE\_CAT*, *PKTABLE\_SCHEM*, *PKTABLE\_NAME*, and *ORDINAL\_POSITION*.

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

Table 64. Columns returned by *SQLForeignKeys*

Column number/name	Data type	Description
1 <i>PKTABLE_CAT</i>	VARCHAR(128)	The current server.
2 <i>PKTABLE_SCHEM</i>	VARCHAR(128)	The name of the schema containing <i>PKTABLE_NAME</i> .
3 <i>PKTABLE_NAME</i>	VARCHAR(128) not NULL	Name of the table containing the primary key.
4 <i>PKCOLUMN_NAME</i>	VARCHAR(128) not NULL	Primary key column name.
5 <i>FKTABLE_CAT</i>	VARCHAR(128)	The current server.

Table 64. Columns returned by SQLForeignKeys (continued)

Column number/name	Data type	Description
6 FKTABLE_SCHEM	VARCHAR(128)	The name of the schema containing FKTABLE_NAME.
7 FKTABLE_NAME	VARCHAR(128) not NULL	The name of the table containing the Foreign key.
8 FKCOLUMN_NAME	VARCHAR(128) not NULL	Foreign key column name.
9 ORDINAL_POSITION	SMALLINT not NULL	The ordinal position of the column in the key, starting at 1.
10 UPDATE_RULE	SMALLINT	Action to be applied to the foreign key when the SQL operation is UPDATE: <ul style="list-style-type: none"> <li>• SQL_RESTRICT</li> <li>• SQL_NO_ACTION</li> </ul> <p>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:</p> <ul style="list-style-type: none"> <li>• SQL_CASCADE</li> <li>• SQL_SET_NULL</li> </ul>
11 DELETE_RULE	SMALLINT	Action to be applied to the foreign key when the SQL operation is DELETE: <ul style="list-style-type: none"> <li>• SQL_CASCADE</li> <li>• SQL_NO_ACTION</li> <li>• SQL_RESTRICT</li> <li>• SQL_SET_DEFAULT</li> <li>• SQL_SET_NULL</li> </ul>
12 FK_NAME	VARCHAR(128)	Foreign key identifier. NULL if not applicable to the data source.
13 PK_NAME	VARCHAR(128)	Primary key identifier. NULL if not applicable to the data source.

**Note:** The column names used by DB2 UDB 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 65. SQLForeignKeys SQLSTATEs

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	A cursor is already opened on the statement handle.
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.

Table 65. *SQLForeignKeys SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	The arguments <i>PKTableName</i> and <i>FKTableName</i> were both NULL pointers.
HY010	Function sequence error	
HY014	No more handles	DB2 UDB CLI is unable to allocate a handle due to internal resources.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value of one of the name length arguments is 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. Refer to “SQLGetInfo - Get general information” on page 127.
HYC00	Driver not capable	DB2 UDB CLI does not support <i>catalog</i> as a qualifier for table name.
HYT00	Timeout expired	

## Restrictions

None.

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/* From CLI sample browser.c */
/* ... */
SQLRETURN list_foreign_keys( SQLHANDLE hstmt,
                            SQLCHAR * schema,
                            SQLCHAR * tablename
                            ) {

/* ... */
    rc = SQLForeignKeys(hstmt, NULL, 0,
                       schema, SQL_NTS, tablename, SQL_NTS,
                       NULL, 0,
                       NULL, SQL_NTS, NULL, SQL_NTS);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) pktable_schem.s, 129,
                   &pktable_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) pktable_name.s, 129,
                   &pktable_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) pkcolumn_name.s, 129,
                   &pkcolumn_name.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 6, SQL_C_CHAR, (SQLPOINTER) fhtable_schem.s, 129,
                   &fhtable_schem.ind);
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

    rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) fhtable_name.s, 129,

```

```

        &fktable_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 8, SQL_C_CHAR, (SQLPOINTER) fkcolumn_name.s, 129,
        &fkcolumn_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 10, SQL_C_SHORT, (SQLPOINTER) &update_rule,
        0, &update_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 11, SQL_C_SHORT, (SQLPOINTER) &delete_rule,
        0, &delete_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 12, SQL_C_CHAR, (SQLPOINTER) fkey_name.s, 129,
        &fkey_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

rc = SQLBindCol(hstmt, 13, SQL_C_CHAR, (SQLPOINTER) pkey_name.s, 129,
        &pkey_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

printf("Primary Key and Foreign Keys for %s.%s\n", schema, tablename);
/* Fetch each row, and display */
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    printf(" %s %s.%s.%s\n      Update Rule ",
        pkcolumn_name.s, fktable_schem.s, fktable_name.s, fkcolumn_name.s);
    if (update_rule == SQL_RESTRIC) {
        printf("RESTRICT "); /* always for IBM DBMSs */
    } else {
        if (update_rule == SQL_CASCADE) {
            printf("CASCADE "); /* non-IBM only */
        } else {
            printf("SET NULL ");
        }
    }
    printf(", Delete Rule: ");
    if (delete_rule == SQL_RESTRIC) {
        printf("RESTRICT "); /* always for IBM DBMSs */
    } else {
        if (delete_rule == SQL_CASCADE) {
            printf("CASCADE "); /* non-IBM only */
        } else {
            if (delete_rule == SQL_NO_ACTION) {
                printf("NO ACTION "); /* non-IBM only */
            } else {
                printf("SET NULL ");
            }
        }
    }
    printf("\n");
    if (pkey_name.ind > 0 ) {
        printf("      Primary Key Name: %s\n", pkey_name.s);
    }
    if (fkey_name.ind > 0 ) {
        printf("      Foreign Key Name: %s\n", fkey_name.s);
    }
}
}

```

## References

- "SQLPrimaryKeys - Get primary key columns of a table" on page 168
- "SQLStatistics - Get index and statistics information for a base table" on page 209



## SQLFreeConnect - Free connection handle

### Purpose

SQLFreeConnect() invalidates and frees the connection handle. All DB2 UDB CLI resources associated with the connection handle are freed.

SQLDisconnect() must be called before calling this function.

Either SQLFreeEnv() is called next to continue terminating the application, or SQLAllocHandle(), to allocate a new connection handle.

### Syntax

```
SQLRETURN SQLFreeConnect (SQLHDBC hdbc);
```

### Function arguments

Table 66. SQLFreeConnect arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Connection handle

### Usage

If this function is called when a connection still exists, SQL\_ERROR is returned, and the connection handle remains valid.

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 67. SQLFreeConnect SQLSTATEs

SQLSTATE	Description	Explanation
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	The function is called before SQLDisconnect() for the <i>hdbc</i> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

### Example

Refer to the example in SQLAllocEnv().

### References

- “SQLDisconnect - Disconnect from a data source” on page 73
- “SQLFreeEnv - Free environment handle” on page 100

# SQLFreeEnv - Free environment handle

## Purpose

SQLFreeEnv() invalidates and frees the environment handle. All DB2 UDB CLI resources associated with the environment handle are freed.

SQLFreeConnect() must be called before calling this function.

This function is the last DB2 UDB CLI step an application needs before terminating.

## Syntax

```
SQLRETURN SQLFreeEnv (SQLHENV henv);
```

## Function arguments

Table 68. SQLFreeEnv arguments

Data type	Argument	Use	Description
SQLHENV	henv	Input	Environment handle

## Usage

If this function is called when there is still a valid connection handle, SQL\_ERROR is returned, and the environment handle remains valid.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 69. SQLFreeEnv SQLSTATEs

SQLSTATE	Description	Explanation
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	There is an <i>hdbc</i> which is in allocated or connected state. Call <i>SQLDisconnect</i> and <i>SQLFreeConnect</i> for the <i>hdbc</i> before calling <i>SQLFreeEnv</i> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## Example

Refer to the example in *SQLAllocEnv()*.

## References

“SQLFreeConnect - Free connection handle” on page 99

## SQLFreeHandle - Free a handle

### Purpose

SQLFreeHandle() invalidates and frees a handle.

### Syntax

```
SQLRETURN SQLFreeHandle (SQLSMALLINT htype,  
                        SQLINTEGER handle);
```

### Function arguments

Table 70. SQLFreeHandle arguments

Data type	Argument	Use	Description
SQLINTEGER	<i>handle</i>	Input	The handle to be freed.
SQLSMALLINT	<i>hType</i>	Input	Handle type that must be SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, or SQL_HANDLE_DESC.

### Usage

SQLFreeHandle() combines the function of SQLFreeEnv(), SQLFreeConnect(), and SQLFreeStmt().

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 71. SQLFreeHandle SQLSTATEs

SQLSTATE	Description	Explanation
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	There is an <i>hdbc</i> which is in allocated or connected state. Call SQLDisconnect and SQLFreeConnect for the <i>hdbc</i> before calling SQLFreeHandle.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

### References

- “SQLFreeConnect - Free connection handle” on page 99
- “SQLFreeEnv - Free environment handle” on page 100
- “SQLFreeStmt - Free (or reset) a statement handle” on page 102

# SQLFreeStmt - Free (or reset) a statement handle

## Purpose

SQLFreeStmt() ends processing on the statement referenced by the statement handle. Use this function to:

- Close a cursor
- Reset parameters
- Unbind columns from variables
- Drop the statement handle and free the DB2 UDB CLI resources associated with the statement handle.

SQLFreeStmt() is called after executing an SQL statement and processing the results.

## Syntax

```
SQLRETURN SQLFreeStmt (SQLHSTMT      hstmt,  
                      SQLSMALLINT   fOption);
```

## Function arguments

Table 72. SQLFreeStmt arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle
SQLSMALLINT	<i>fOption</i>	Input	Option specifying the manner of freeing the statement handle. The option must have one of the following values: <ul style="list-style-type: none"><li>• SQL_CLOSE</li><li>• SQL_DROP</li><li>• SQL_UNBIND</li><li>• SQL_RESET_PARAMS</li></ul>

## Usage

SQLFreeStmt() can be called with the following options:

- SQL\_CLOSE

The cursor (if any) associated with the statement handle (*hstmt*) 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 *hstmt*. The cursor name is retained until the statement handle is dropped or the next successful SQLSetCursorName() call. If no cursor has been associated with the statement handle, this option has no effect (no warning or error is generated).

- SQL\_DROP

DB2 UDB 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.

- SQL\_UNBIND

All the columns bound by previous SQLBindCol() calls on this statement handle are released (the association between application variables or file references and result set columns is broken).

- SQL\_RESET\_PARAMS

All the parameters set by previous SQLBindParam() calls on this statement handle are released. The association between application variables or file references and parameter markers in the SQL statement of the statement handle is broken.

To reuse a statement handle to run a different statement and if the previous statement:

- Was a SELECT, you must close the cursor.
- Used a different number or type of parameters, the parameters must be reset.
- Used a different number or type of column bindings, the columns must be unbound.

Alternatively you can drop the statement handle and allocate a new one.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_IN\_HANDLE

SQL\_SUCCESS\_WITH\_INFO is not returned if *fOption* is set to SQL\_DROP, because there is no statement handle to use when SQLError() is called.

## Diagnostics

Table 73. SQLFreeStmt SQLSTATES

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The value specified for the argument <i>fOption</i> is not SQL_CLOSE, SQL_DROP, SQL_UNBIND, or SQL_RESET_PARAMS.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## Example

Refer to the SQLFetch() "" on page 88.

## References

- "SQLAllocStmt - Allocate a statement handle" on page 27
- "SQLBindCol - Bind a column to an application variable" on page 29
- "SQLFetch - Fetch next row" on page 87
- "SQLFreeConnect - Free connection handle" on page 99
- "SQLSetParam - Set parameter" on page 200

## SQLGetCol - Retrieve one column of a row of the result set

### Purpose

SQLGetCol() retrieves data for a single column in the current row of the result set. This is an alternative to SQLBindCol(), which transfers data directly into application variables on a call to SQLFetch(). SQLGetCol() is also used to retrieve large character based data in pieces.

SQLFetch() must be called before SQLGetCol().

After calling SQLGetCol() for each column, SQLFetch() is called to retrieve the next row.

### Syntax

```
SQLRETURN SQLGetCol (SQLHSTMT      hstmt,
                    SQLSMALLINT    icol,
                    SQLSMALLINT    fcType,
                    SQLPOINTER     rgbValue,
                    SQLINTEGER     cbValueMax,
                    SQLINTEGER     *pcbValue);
```

### Function arguments

Table 74. SQLGetCol arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER *	<i>pcbValue</i>	Output	<p>Pointer to the value that indicates the number of bytes DB2 UDB CLI has available to return in the <i>rgbValue</i> buffer. If the data is being retrieved in pieces, this contains the number of bytes still remaining, excluding any bytes of the column's data that has been obtained from previous calls to SQLGetCol().</p> <p>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 fails because it has no means of reporting this.</p> <p>If SQLFetch() has fetched a column containing graphic data, then the pointer to <i>pcbValue</i> must not be NULL or this function fails because it has no means of informing the application about the length of the data retrieved in the <i>rgbValue</i> buffer.</p>
SQLINTEGER	<i>cbValueMax</i>	Input	Maximum size of the buffer pointed to by <i>rgbValue</i> . If <i>fcType</i> is either SQL_DECIMAL or SQL_NUMERIC, <i>cbValueMax</i> must be a precision and scale. The method to specify both values is to use (precision * 256) + scale. This is also the value returned as the LENGTH of these data types when using SQLColAttributes().
SQLPOINTER	<i>rgbValue</i>	Output	Pointer to buffer where the retrieved column data is to be stored.

Table 74. SQLGetCol arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>fCType</i>	Input	Application data type of the column identified by <i>icol</i> . The following types are supported: <ul style="list-style-type: none"> <li>• SQL_BIGINT</li> <li>• SQL_BINARY</li> <li>• SQL_BLOB</li> <li>• SQL_CHAR</li> <li>• SQL_CLOB</li> <li>• SQL_DATETIME</li> <li>• SQL_DBCLOB</li> <li>• SQL_DECIMAL</li> <li>• SQL_DOUBLE</li> <li>• SQL_FLOAT</li> <li>• SQL_GRAPHIC</li> <li>• SQL_INTEGER</li> <li>• SQL_NUMERIC</li> <li>• SQL_REAL</li> <li>• SQL_SMALLINT</li> <li>• SQL_TYPE_DATE</li> <li>• SQL_TYPE_TIME</li> <li>• SQL_TYPE_TIMESTAMP</li> <li>• SQL_VARBINARY</li> <li>• SQL_VARCHAR</li> <li>• SQL_VARGRAPHIC</li> </ul>
SQLSMALLINT	<i>icol</i>	Input	Column number for which the data retrieval is requested.

## Usage

SQLGetCol() can be used with SQLBindCol() for the same row, as long as the value of *icol* does not specify a column that has been bound. The general steps are:

1. SQLFetch() - advances cursor to first row, retrieves first row, transfers data for bound columns.
2. SQLGetCol() - transfers data for specified (unbound) column.
3. Repeat step 2 for each column needed.
4. SQLFetch() - advances cursor to next row, retrieves next row, transfers data for bound columns.
5. Repeat steps 2, 3 and 4 for each row in the result set, or until the result set is no longer needed.

SQLGetCol() retrieves long columns if the C data type (*fCType*) is SQL\_CHAR or if *fCType* is SQL\_DEFAULT and the column type is CHAR or VARCHAR.

On each SQLGetCol() call, if the data available for return is greater than or equal to *cbValueMax*, truncation occurs. A function return code of SQL\_SUCCESS\_WITH\_INFO that is coupled with an SQLSTATE that denotes data truncation indicates truncation. The application can call SQLGetCol() again, with the same *icol* value, to obtain later 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 SQLGetCol() returns SQL\_NO\_DATA\_FOUND.

To discard the column data part way through the retrieval, the application can call `SQLGetCol()` with *icol* set to the next column position of interest. To discard unretrieved data for the entire row, the application should call `SQLFetch()` to advance the cursor to the next row; or, if it is not interested in any more data from the result set, call `SQLFreeStmt()` to close the cursor.

The *fCType* input argument determines the type of data conversion (if any) needed before the column data is placed into the storage area pointed to by *rgbValue*.

The contents returned in *rgbValue* is always null-terminated unless `SQLSetEnvAttr()` is used to change the `SQL_ATTR_OUTPUT_NTS` attribute or if the application is retrieving the data in multiple chunks. If the application is retrieving the data in multiple chunks, the null-terminating byte is only added to the last portion of data.

Truncation of numeric data types is not reported 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).

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

`SQL_NO_DATA_FOUND` is returned when the preceding `SQLGetCol()` call has retrieved all of the data for this column.

`SQL_SUCCESS` is returned if a zero-length string is retrieved by `SQLGetCol()`. If this is the case, *pcbValue* contains 0, and *rgbValue* contains a null terminator.

If the preceding call to `SQLFetch()` fails, `SQLGetCol()` should not be called because the result is undefined.

## Diagnostics

Table 75. *SQLGetCol* SQLSTATES

SQLSTATE	Description	Explanation
07006	Restricted data type attribute violation	The data value cannot be converted to the C data type specified by the argument <i>fCType</i> .
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The value of the argument <i>cbValueMax</i> is less than 1 and the argument <i>fCType</i> is <code>SQL_CHAR</code> .  The specified column number is not valid.  The argument <i>rgbValue</i> or <i>pcbValue</i> is a null pointer.
HY010	Function sequence error	The specified <i>hstmt</i> is not in a cursor positioned state. The function is called without first calling <code>SQLFetch()</code> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.



Table 75. SQLGetCol SQLSTATEs (continued)

SQLSTATE	Description	Explanation
HYC00	Driver not capable	The SQL data type for the specified data type is recognized but not supported by the driver.  The requested conversion from the SQL data type to the application data <i>fCType</i> cannot be performed by the driver or the data source.

## Restrictions

ODBC requires that *icol* not specify a column of a lower number than the column last retrieved by SQLGetCol() for the same row on the same statement handle. ODBC also does not permit the use of SQLGetCol() to retrieve data for a column that resides before the last bound column, (if any columns in the row have been bound).

DB2 UDB CLI has relaxed both of these rules by allowing the value of *icol* to be specified in any order and before a bound column, provided that *icol* does not specify a bound column.

## Example

Refer to the example in the “SQLFetch - Fetch next row” on page 87 for a comparison between using bound columns and using SQLGetCol().

Refer to “Example: Interactive SQL and the equivalent DB2 UDB CLI function calls” on page 253 for a listing of the check\_error, initialize, and terminate functions used in the following example.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/*****
** file = getcol.c
**
** Example of directly executing an SQL statement.
** Getcol is used to retrieve information from the result set.
** Compare to fetch.c
**
** Functions used:
**
**      SQLAllocConnect      SQLFreeConnect
**      SQLAllocEnv         SQLFreeEnv
**      SQLAllocStmt        SQLFreeStmt
**      SQLConnect          SQLDisconnect
**
**      SQLBindCol          SQLFetch
**      SQLTransact         SQLError
**      SQLExecDirect       SQLGetCursor
*****/

#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
              SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

```

```

int print_error (SQLHENV   henv,
                SQLHDBC   hdbc,
                SQLHSTMT  hstmt);

int check_error (SQLHENV   henv,
                SQLHDBC   hdbc,
                SQLHSTMT  hstmt,
                SQLRETURN  frc);

/*****
** main
** - initialize
** - terminate
*****/
int main()
{
    SQLHENV   henv;
    SQLHDBC   hdbc;
    SQLCHAR   sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN rc;

    rc = initialize(&henv, &hdbc);
    if (rc != SQL_SUCCESS) return(terminate(henv, hdbc));

    {SQLHSTMT  hstmt;
    SQLCHAR   sqlstmt []="SELECT deptname, location from org where division = 'Eastern'";
    SQLCHAR   deptname[15],
              location[14];
    SQLINTEGER rlength;

        rc = SQLAllocStmt(hdbc, &hstmt);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

        rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS);
        if (rc != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt, rc);

        printf("Departments in Eastern division:\n");
        printf("DEPTNAME      Location\n");
        printf("-----\n");

        while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
        {
            rc = SQLGetCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15, &rlength);
            rc = SQLGetCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14, &rlength);
            printf("%-14.14s %-13.13s \n", deptname, location);
        }
        if (rc != SQL_NO_DATA_FOUND )
            check_error (henv, hdbc, hstmt, rc);
    }

    rc = SQLTransact(henv, hdbc, SQL_COMMIT);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    terminate(henv, hdbc);
    return (SQL_SUCCESS);
}
/* end main */

```

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLFetch - Fetch next row” on page 87

# SQLGetConnectAttr - Get the value of a connection attribute

## Purpose

SQLGetConnectAttr() returns the current settings for the specified connection option.

These options are set using the SQLSetConnectAttr() function.

## Syntax

```
SQLRETURN SQLGetConnectAttr( SQLHDBC hdbc,
                             SQLINTEGER fAttr,
                             SQLPOINTER pvParam), ;
                             SQLINTEGER bLen,
                             SQLINTEGER *sLen);
```

## Function arguments

Table 76. SQLGetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Connection handle.
SQLINTEGER *	<i>sLen</i>	Output	Length of the output data, if the attribute is a character string; otherwise, unused.
SQLINTEGER	<i>bLen</i>	Input	Maximum number of bytes to store in <i>pvParam</i> , if the value is a character string; otherwise, unused.
SQLINTEGER	<i>fAttr</i>	Input	Attribute to retrieve. See "SQLSetConnectAttr - Set a connection attribute" on page 183 for a description of the connect options.
SQLPOINTER	<i>pvParam</i>	Output	Value associated with <i>fAttr</i> . Depending on the value of <i>fAttr</i> . This can be a 32-bit integer value, or a pointer to a null terminated character string.

## Usage

If SQLGetConnectAttr() is called, and the specified *fAttr* has not been set through SQLSetConnectAttr and does not have a default, then SQLGetConnectAttr() returns SQL\_NO\_DATA\_FOUND.

Statement options settings cannot be retrieved through SQLGetConnectAttr().

## Diagnostics

Table 77. SQLGetConnectAttr SQLSTATEs

SQLSTATE	Description	Explanation
08003	Connection not open	An <i>fAttr</i> value that requires an open connection is specified .
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Attribute type out of range	An <i>fAttr</i> value that is not valid is specified. The argument <i>pvParam</i> is a null pointer.
HYC00	Driver not capable	The <i>fAttr</i> argument is recognized, but is not supported.

## SQLGetConnectOption - Return current setting of a connect Option

### Purpose

SQLGetConnectOption() has been deprecated and replaced with SQLGetConnectAttr(). Although this version of DB2 UDB CLI continues to support SQLGetConnectOption(), it is recommended that you begin using SQLGetConnectAttr() in your DB2 UDB CLI programs so that they conform to the latest standards.

SQLGetConnectOption() returns the current settings for the specified connection option.

These options are set using the SQLSetConnectOption() function.

### Syntax

```
SQLRETURN SQLGetConnectOption( HDBC          hdbc,  
                               SQLSMALLINT  fOption,  
                               SQLPOINTER   pvParam);
```

### Function arguments

Table 78. SQLGetConnectOption arguments

Data type	argument	Use	Description
HDBC	<i>hdbc</i>	Input	Connection handle.
SQLPOINTER	<i>pvParam</i>	Output	Value associated with <i>fOption</i> . Depending on the value of <i>fOption</i> , this can be a 32-bit integer value, or a pointer to a null terminated character string. The maximum length of any character string returned is <code>SQL_MAX_OPTION_STRING_LENGTH</code> bytes (excluding the null-terminating byte).
SQLSMALLINT	<i>fOption</i>	Input	Option to retrieve. Refer to Table 146 on page 184 for more information.

### Usage

SQLGetConnectOption() provides the same function as SQLGetConnectAttr(), both functions are supported for compatibility reasons.

If SQLGetConnectOption() is called, and the specified *fOption* has not been set through SQLSetConnectOption and does not have a default, then SQLGetConnectOption() returns `SQL_NO_DATA_FOUND`.

Statement options settings cannot be retrieved through SQLGetConnectOption().

### Diagnostics

Table 79. SQLGetConnectOption SQLSTATEs

SQLSTATE	Description	Explanation
08003	Connection not open	An <i>fOption</i> value that requires an open connection is specified .
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.

Table 79. *SQLGetConnectOption SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
HY009	Option type out of range	An <i>fOption</i> value that is not valid is specified.  The argument <i>pvParam</i> is a null pointer.
HYC00	Driver not capable	The <i>fOption</i> argument is recognized, but is not supported.

## SQLGetCursorName - Get cursor name

### Purpose

SQLGetCursorName() returns the cursor name associated with the input statement handle. If a cursor name is explicitly set by calling SQLSetCursorName(), this name is returned; otherwise, an implicitly generated name is returned.

### Syntax

```
SQLRETURN SQLGetCursorName (SQLHSTMT    hstmt,
                             SQLCHAR     *szCursor,
                             SQLSMALLINT cbCursorMax,
                             SQLSMALLINT *pcbCursor);
```

### Function arguments

Table 80. *SQLGetCursorName arguments*

Data type	Argument	Use	Description
SQLCHAR *	<i>szCursor</i>	Output	Cursor name
SQLHSTMT	<i>hstmt</i>	Input	Statement handle
SQLSMALLINT *	<i>pcbCursor</i>	Output	Amount of bytes available to return for <i>szCursor</i>
SQLSMALLINT	<i>cbCursorMax</i>	Input	Length of buffer <i>szCursor</i>

### Usage

SQLGetCursorName() returns a cursor name if a name is set using SQLSetCursorName() or if a SELECT statement is processed on the statement handle. If neither of these is true, then calling SQLGetCursorName() results in an error.

If a name is set explicitly using SQLSetCursorName(), this name is returned until the statement is dropped, or until another explicit name is set.

If an explicit name is not set, an implicit name is generated when a SELECT statement is processed, and this name is returned. Implicit cursor names always begin with SQLCUR.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 81. SQLGetCursorName SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated	The cursor name returned in <i>szCursor</i> is longer than the value in <i>cbCursorMax</i> , and is truncated to <i>cbCursorMax</i> - 1 bytes. The argument <i>pcbCursor</i> contains the length of the full cursor name available for return. The function returns SQL_SUCCESS_WITH_INFO.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>szCursor</i> or <i>pcbCursor</i> is a null pointer.  The value specified for the argument <i>cbCursorMax</i> is less than 1.
HY010	Function sequence error	The statement <i>hstmt</i> is not in execute state. Call SQLExecute(), SQLExecDirect() or SQLSetCursorName() before calling SQLGetCursorName().
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY015	No cursor name available.	There is no open cursor on the <i>hstmt</i> and no cursor name has been set with SQLSetCursorName(). The statement associated with <i>hstmt</i> does not support the use of a cursor.

## Restrictions

ODBC's generated cursor names start with SQL\_CUR and X/Open CLI generated cursor names begin with SQLCUR. DB2 UDB CLI uses SQLCUR.

## Example

Refer to "Example: Interactive SQL and the equivalent DB2 UDB CLI function calls" on page 253 for a listing of the check\_error, initialize, and terminate functions used in the following example.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 260.

```

/*****
** file = getcurs.c
**
** Example of directly executing a SELECT and positioned UPDATE SQL statement.
** Two statement handles are used, and SQLGetCursor is used to retrieve the
** generated cursor name.
**
** Functions used:
**
**      SQLAllocConnect      SQLFreeConnect
**      SQLAllocEnv         SQLFreeEnv
**      SQLAllocStmt        SQLFreeStmt
**      SQLConnect          SQLDisconnect
**
**      SQLBindCol          SQLFetch
**      SQLTransact         SQLError

```

```

**          SQLExecDirect          SQLGetCursorName
*****/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,
              SQLHDBC *hdbc);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error (SQLHENV    henv,
                 SQLHDBC   hdbc,
                 SQLHSTMT  hstmt);

int check_error (SQLHENV    henv,
                 SQLHDBC   hdbc,
                 SQLHSTMT  hstmt,
                 SQLRETURN  frc);

/*****
** main
** - initialize
** - terminate
*****/
int main()
{
    SQLHENV    henv;
    SQLHDBC    hdbc;
    SQLRETURN  rc,
              rc2;

    rc = initialize(&henv, &hdbc);
    if (rc != SQL_SUCCESS) return(terminate(henv, hdbc));

    {SQLHSTMT  hstmt1,
      hstmt2;
     SQLCHAR  sqlstmt[]="SELECT name, job from staff for update of job";
     SQLCHAR  updstmt[MAX_STMT_LEN + 1];
     SQLCHAR  name[10],
              job[6],
              newjob[6],
              cursor[19];

     SQLINTEGER  rlength, attr;
     SQLSMALLINT clength;

     rc = SQLAllocStmt(hdbc, &hstmt1);
     if (rc != SQL_SUCCESS )
         check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

     /* make sure the statement is update-capable */
     attr = SQL_FALSE;
     rc = SQLSetStmtAttr(hstmt1,SQL_ATTR_FOR_FETCH_ONLY, &attr, 0);

     /* allocate second statement handle for update statement */
     rc2 = SQLAllocStmt(hdbc, &hstmt2);
     if (rc2 != SQL_SUCCESS )
         check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

     rc = SQLExecDirect(hstmt1, sqlstmt, SQL_NTS);
     if (rc != SQL_SUCCESS )
         check_error (henv, hdbc, hstmt1, rc);

```

```

/* Get Cursor of the SELECT statement's handle */
rc = SQLGetCursorName(hstmt1, cursor, 19, &clength);
if (rc != SQL_SUCCESS )
    check_error (henv, hdbc, hstmt1, rc);

/* bind name to first column in the result set */
rc = SQLBindCol(hstmt1, 1, SQL_CHAR, (SQLPOINTER) name, 10,
                &rlength);
if (rc != SQL_SUCCESS )
    check_error (henv, hdbc, hstmt1, rc);

/* bind job to second column in the result set */
rc = SQLBindCol(hstmt1, 2, SQL_CHAR, (SQLPOINTER) job, 6,
                &rlength);
if (rc != SQL_SUCCESS )
    check_error (henv, hdbc, hstmt1, rc);

printf("Job Change for all clerks\n");

while ((rc = SQLFetch(hstmt1)) == SQL_SUCCESS)
{
    printf("Name: %-9.9s Job: %-5.5s \n", name, job);
    printf("Enter new job or return to continue\n");
    gets(newjob);
    if (newjob[0] != '\0')
    {
        sprintf( updstmt,
                "UPDATE staff set job = '%s' where current of %s",
                newjob, cursor);
        rc2 = SQLExecDirect(hstmt2, updstmt, SQL_NTS);
        if (rc2 != SQL_SUCCESS )
            check_error (henv, hdbc, hstmt2, rc);
    }
}
if (rc != SQL_NO_DATA_FOUND )
    check_error (henv, hdbc, hstmt1, rc);
SQLFreeStmt(hstmt1, SQL_CLOSE);
}

printf("Committing Transaction\n");
rc = SQLTransact(henv, hdbc, SQL_COMMIT);
if (rc != SQL_NO_DATA_FOUND )
    check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

terminate(henv, hdbc);
return (0);
}/* end main */

```

## References

- "SQLExecute - Execute a statement" on page 83
- "SQLExecDirect - Execute a statement directly" on page 81
- "SQLSetCursorName - Set cursor name" on page 192

## SQLGetData - Get data from a column

### Purpose

SQLGetData() retrieves data for a single column in the current row of the result set. This is an alternative to SQLBindCol(), which transfers data directly into application variables on a call to SQLFetch(). SQLGetData() can also be used to retrieve large character based data in pieces.



SQLFetch() must be called before SQLGetData().

After calling SQLGetData() for each column, SQLFetch() is called to retrieve the next row.

SQLGetData() is identical to SQLGetCol(), both functions are supported for compatibility reasons.

### Syntax

```
SQLRETURN SQLGetData (SQLHSTMT      hstmt,
                    SQLSMALLINT    icol,
                    SQLSMALLINT    fCType,
                    SQLPOINTER     rgbValue,
                    SQLINTEGER     cbValueMax,
                    SQLINTEGER     *pcbValue);
```

**Note:** Refer to “SQLGetCol - Retrieve one column of a row of the result set” on page 103 for a description of the applicable sections.

## SQLGetDescField - Get descriptor field

### Purpose

SQLGetDescField() obtains a value from a descriptor. SQLGetDescField() is a more extensible alternative to the SQLGetDescRec() function.

This function is similar to that of SQLDescribeCol() but SQLGetDescField() can retrieve data from parameter descriptors as well as row descriptors.

### Syntax

```
SQLRETURN SQLGetDescField (SQLHDESC   hdesc,
                          SQLSMALLINT irec,
                          SQLSMALLINT fDescType,
                          SQLPOINTER  rgbDesc,
                          SQLINTEGER  bLen,
                          SQLINTEGER  *sLen);
```

### Function arguments

Table 82. SQLGetDescField arguments

Data type	Argument	Use	Description
SQLHDESC	<i>hdesc</i>	Input	Descriptor handle.
SQLSMALLINT	<i>irec</i>	Input	The number of records in the descriptor matches the number of columns in the result set for a row descriptor, or the number of parameters in a parameter descriptor.
SQLSMALLINT	<i>fDescType</i>	Input	See Table 83 on page 116.
SQLPOINTER	<i>rgbDesc</i>	Output	Pointer to buffer.
SQLINTEGER	<i>bLen</i>	Input	Length of descriptor buffer ( <i>rgbDesc</i> ).
SQLINTEGER *	<i>sLen</i>	Output	Actual number of bytes in the descriptor to return. If this argument contains a value equal to or higher than the length <i>rgbDesc</i> buffer, truncation occurs.

Table 83. *fDescType* descriptor types

Descriptor	Type	Description
SQL_DESC_ALLOC_TYPE	SMALLINT	Either SQL_DESC_ALLOC_USER if the application explicitly allocated the descriptor, or SQL_DESC_ALLOC_AUTO if the implementation automatically allocated the descriptor.
SQL_DESC_COUNT	SMALLINT	The number of records in the descriptor is returned in <i>rgbDesc</i> .
SQL_DESC_DATA_PTR	SQLPOINTER	Retrieve the data pointer field for <i>irec</i> .
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	Retrieve the interval code for records with a type of SQL_DATETIME. The interval code further defines the SQL_DATETIME data type. The code values are SQL_CODE_DATE, SQL_CODE_TIME, and SQL_CODE_TIMESTAMP.
SQL_DESC_INDICATOR_PTR	SQLPOINTER	Retrieve the indicator pointer field for <i>irec</i> .
SQL_DESC_LENGTH_PTR	SQLPOINTER	Retrieve the length pointer field for <i>irec</i> .
SQL_DESC_LENGTH	INTEGER	Retrieve the LENGTH field of <i>irec</i> .
SQL_DESC_NAME	CHAR(128)	Retrieve the NAME field of <i>irec</i> .
SQL_DESC_NULLABLE	SMALLINT	If <i>irec</i> can contain nulls, then SQL_NULLABLE is returned in <i>rgbDesc</i> . Otherwise, SQL_NO_NULLS is returned in <i>rgbDesc</i> .
SQL_DESC_PRECISION	SMALLINT	Retrieve the PRECISION field of <i>irec</i> .
SQL_DESC_SCALE	SMALLINT	Retrieve the SCALE field of <i>irec</i> .
SQL_DESC_TYPE	SMALLINT	Retrieve the TYPE field of <i>irec</i> .
SQL_DESC_UNNAMED	SMALLINT	This is SQL_NAMED if the NAME field is an actual name, or SQL_UNNAMED if the NAME field is an implementation-generated name.

## Usage

The number of records in the descriptor corresponds to the number of columns in the result set, if the descriptor is row descriptor, or the number of parameters, for a parameter descriptor.

Calling `SQLGetDescField()` with *fDescType* set to `SQL_DESC_COUNT` is an alternative to calling `SQLNumResultCols()` to determine whether any columns can be returned.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

## Diagnostics

Table 84. SQLGetDescField SQLSTATEs

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	The value specified for the argument <i>fDescType</i> or <i>irec</i> is not valid.  The argument <i>rgbDesc</i> or <i>sLen</i> is a null pointer.
HY013 *	Memory management problem	The driver is unable to access the memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLPrepare - Prepare a statement” on page 164

## SQLGetDescRec - Get descriptor record

### Purpose

SQLGetDescRec() obtains an entire record from a descriptor. SQLGetDescRec() is a more concise alternative to the SQLGetDescField() function.

### Syntax

```
SQLRETURN SQLGetDescRec (SQLHDESC      hdesc,
                        SQLSMALLINT    irec,
                        SQLCHAR        *rgbDesc,
                        SQLSMALLINT    cbDescMax,
                        SQLSMALLINT    *pcbDesc,
                        SQLSMALLINT    *type,
                        SQLSMALLINT    *subtype,
                        SQLINTEGER     *length,
                        SQLSMALLINT    *prec,
                        SQLSMALLINT    *scale,
                        SQLSMALLINT    *nullable);
```

### Function arguments

Table 85. SQLGetDescRec arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>rgbDesc</i>	Output	NAME field for the record.
SQLHDESC	<i>hdesc</i>	Input	Descriptor handle.
SQLINTEGER *	<i>length</i>	Output	LENGTH field for the record.
SQLSMALLINT *	<i>nullable</i>	Output	NULLABLE field for the record.

Table 85. *SQLGetDescRec* arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT *	<i>pcbDesc</i>	Output	Total length of the output data.
SQLSMALLINT *	<i>prec</i>	Output	PRECISION field for the record.
SQLSMALLINT *	<i>scale</i>	Output	SCALE field for the record.
SQLSMALLINT *	<i>subtype</i>	Output	DATETIME_INTERVAL_CODE, for records whose TYPE is SQL_DATETIME.
SQLSMALLINT *	<i>type</i>	Output	TYPE field for the record.
SQLSMALLINT	<i>cbDescMax</i>	Input	Maximum number of bytes to store in <i>rgbDesc</i> .
SQLSMALLINT	<i>irec</i>	Input	The number of records in the descriptor matches the number of columns in the result set for a row descriptor, or the number of parameters in a parameter descriptor.

## Usage

Calling `SQLGetDescRec()` retrieves all the data from a descriptor record in one call. It might still be necessary to call `SQLGetDescField()` with `SQL_DESC_COUNT` to determine the number of records in the descriptor.

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

## Diagnostics

Table 86. *SQLGetDescRec* SQLSTATEs

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	The value specified for the argument <i>irec</i> is not valid.  The argument <i>rgbDesc</i> , <i>pcbDesc</i> , <i>type</i> , <i>subtype</i> , <i>length</i> , <i>prec</i> , <i>scale</i> or <i>nullable</i> is a null pointer.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLPrepare - Prepare a statement” on page 164

## SQLGetDiagField - Return diagnostic information (extensible)

### Purpose

SQLGetDiagField() returns the diagnostic information associated with the most recently called DB2 UDB CLI function for a particular statement, connection or environment handle.

The information consists of a standardized SQLSTATE, an error code, and a text message.

Call SQLGetDiagField() after receiving a return code of SQL\_ERROR or SQL\_SUCCESS\_WITH\_INFO from another function call.

**Note:** Some database servers might provide product-specific diagnostic information after returning SQL\_NO\_DATA\_FOUND from the processing of a statement.

### Syntax

```
SQLRETURN SQLGetDiagField (SQLSMALLINT    htype,
                          SQLINTEGER      handle,
                          SQLSMALLINT    recNum,
                          SQLSMALLINT    diagId,
                          SQLPOINTER     diagInfo,
                          SQLSMALLINT    bLen,
                          SQLSMALLINT    *sLen);
```

### Function arguments

Table 87. SQLDiagField arguments

Data type	Argument	Use	Description
SQLSMALLINT	<i>hType</i>	Input	Handle type.
SQLINTEGER	<i>handle</i>	Input	Handle for which the diagnostic information is wanted.
SQLSMALLINT	<i>recNum</i>	Input	If there are multiple errors, this indicates which one should be retrieved. If header information is requested, this must be 0. The first error record is number 1.
SQLSMALLINT	<i>diagId</i>	Input	See Table 88.
SQLPOINTER	<i>diagInfo</i>	Output	Buffer for diagnostic information.
SQLSMALLINT	<i>bLen</i>	Input	Length of <i>diagInfo</i> , if requested data is a character string; otherwise, unused.
SQLSMALLINT *	<i>sLen</i>	Output	Length of complete diagnostic information, if the requested data is a character string; otherwise, unused.

Table 88. DiagId types

Descriptor	Type	Description
SQL_DIAG_MESSAGE_TEXT	CHAR(254)	The implementation-defined message text relating to the diagnostic record.
SQL_DIAG_NATIVE	INTEGER	The implementation-defined error code relating to the diagnostic record. Portable applications should not base their behavior on this value.
SQL_DIAG_NUMBER	INTEGER	The number of diagnostic records available for the specified handle.

Table 88. *DiagId* types (continued)

Descriptor	Type	Description
SQL_DIAG_RETURNCODE	SMALLINT	Return code of the underlying function. Can be SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA_FOUND, or SQL_ERROR.
SQL_DIAG_ROW_COUNT	INTEGER	The number of rows for the specified handle, if the handle is a statement handle.
SQL_DIAG_SERVER_NAME	CHAR(128)	The server name that the diagnostic record relates to, as it is supplied on the SQLConnect() statement that establishes the connection.
SQL_DIAG_SQLSTATE	CHAR(5)	The 5-character SQLSTATE code relating to the diagnostic record. The SQLSTATE code provides a portable diagnostic indication.

## Usage

The SQLSTATEs are those defined by the X/OPEN SQL CAE and the X/Open SQL CLI snapshot, augmented with SQLSTATE values.

If diagnostic information generated by one DB2 UDB CLI function is not retrieved before a function other than SQLGetDiagField() is called with the same handle, the information for the previous function call is lost. This is true whether diagnostic information is generated for the second DB2 UDB CLI function call.

Multiple diagnostic messages might be available after a given DB2 UDB CLI function call. These messages can be retrieved one at a time by repeatedly calling SQLGetDiagField(). For each message retrieved, SQLGetDiagField() returns SQL\_SUCCESS and removes it from the list of messages available. When there are no more messages to retrieve, SQL\_NO\_DATA\_FOUND is returned.

Diagnostic information stored under a given handle is cleared when a call is made to SQLGetDiagField() with that handle, or when another DB2 UDB CLI function call is made with that handle. However, information associated with a given handle type is not cleared by a call to SQLGetDiagField() with an associated but different handle type. For example, a call to SQLGetDiagField() with a connection handle input does not clear errors associated with any statement handles under that connection.

SQL\_SUCCESS is returned even if the buffer for the error message (*szDiagFieldMsg*) is too short. This is because the application is not able to retrieve the same error message by calling SQLGetDiagField() again. The actual length of the message text is returned in the *pcbDiagFieldMsg*.

To avoid truncation of the error message, declare a buffer length of SQL\_MAX\_MESSAGE\_LENGTH + 1. The message text is never longer than this.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

SQL\_NO\_DATA\_FOUND is returned if no diagnostic information is available for the input handle, or if all of the messages have been retrieved through calls to SQLGetDiagField().

SQL\_ERROR is returned if the argument *diagInfo* or *sLen* is a null pointer.

## Diagnostics

SQLSTATEs are not defined, because SQLGetDiagField() does not generate diagnostic information for itself.

## Restrictions

Although ODBC also returns X/Open SQL CAE SQLSTATEs, only DB2 UDB CLI returns the additional IBM defined SQLSTATEs. The ODBC Driver Manager also returns SQLSTATE values in addition to the standard ones. For more information about ODBC specific SQLSTATEs refer to *Microsoft ODBC Programmer's Reference*.

Because of this, you should only build dependencies on the standard SQLSTATEs. This means any branching logic in the application should only rely on the standard SQLSTATEs. The augmented SQLSTATEs are most useful for debugging purposes.

### Related concepts

“Diagnostics in a DB2 UDB CLI application” on page 14

This topic deals with warning or error conditions generated within an application.

## SQLGetDiagRec - Return diagnostic information (concise)

### Purpose

SQLGetDiagRec() returns the diagnostic information associated with the most recently called DB2 UDB CLI function for a particular statement, connection, or environment handle.

The information consists of a standardized SQLSTATE, the error code, and a text message. Refer to “Diagnostics in a DB2 UDB CLI application” on page 14 for more information.

Call SQLGetDiagRec() after receiving a return code of SQL\_ERROR or SQL\_SUCCESS\_WITH\_INFO from another function call.

**Note:** Some database servers might provide product-specific diagnostic information after returning SQL\_NO\_DATA\_FOUND from the processing of a statement.

### Syntax

```
SQLRETURN SQLGetDiagRec (SQLSMALLINT hType,  
                        SQLINTEGER handle,  
                        SQLSMALLINT recNum,  
                        SQLCHAR *szSqlState,  
                        SQLINTEGER *pfNativeError,  
                        SQLCHAR *szErrorMsg,  
                        SQLSMALLINT cbErrorMsgMax,  
                        SQLSMALLINT *pcbErrorMsg);
```

## Function arguments

Table 89. *SQLGetDiagRec* arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szErrorMsg</i>	Output	Pointer to buffer to contain the implementation defined message text. In DB2 UDB CLI, only the DBMS generated messages are returned; DB2 UDB CLI itself does not return any message text describing the problem.
SQLCHAR *	<i>szSqlState</i>	Output	SQLSTATE as a string of 5 characters terminated by a null character. The first 2 characters indicate error class; the next 3 indicate subclass. The values correspond directly to SQLSTATE values defined in the X/Open SQL CAE specification and the ODBC specification, augmented with IBM specific and product specific SQLSTATE values.
SQLINTEGER *	<i>pfNativeError</i>	Output	Error code. In DB2 UDB CLI, the <i>pfNativeError</i> argument contains the SQLCODE value returned by the Database Management System (DBMS). If the error is generated by DB2 UDB CLI and not the DBMS, then this field is set to -99999.
SQLINTEGER	<i>handle</i>	Input	Handle for which the diagnostic information is wanted.
SQLSMALLINT *	<i>pcbErrorMsg</i>	Output	Pointer to total number of bytes available to return to the <i>szErrorMsg</i> buffer. This does not include the null termination character.
SQLSMALLINT	<i>cbErrorMsgMax</i>	Input	Maximum (that is, the allocated) length of the buffer <i>szErrorMsg</i> . The recommended length to allocate is SQL_MAX_MESSAGE_LENGTH + 1.
SQLSMALLINT	<i>hType</i>	Input	Handle type.
SQLSMALLINT	<i>recNum</i>	Input	If there are multiple errors, this indicates which one should be retrieved. If header information is requested, this must be 0. The first error record is number 1.

## Usage

The SQLSTATEs are those defined by the X/OPEN SQL CAE and the X/Open SQL CLI snapshot, augmented with IBM specific and product specific SQLSTATE values.

If diagnostic information generated by one DB2 UDB CLI function is not retrieved before a function other than `SQLGetDiagRec()` is called with the same handle, the information for the previous function call is lost. This is true whether diagnostic information is generated for the second DB2 UDB CLI function call.

Multiple diagnostic messages might be available after a given DB2 UDB CLI function call. These messages can be retrieved one at a time by repeatedly calling `SQLGetDiagRec()`. For each message retrieved, `SQLGetDiagRec()` returns `SQL_SUCCESS` and removes it from the list of messages available. When there are no more messages to retrieve, `SQL_NO_DATA_FOUND` is returned, the SQLSTATE is set to "00000", *pfNativeError* is set to 0, and *pcbErrorMsg* and *szErrorMsg* are undefined.



Diagnostic information stored under a given handle is cleared when a call is made to `SQLGetDiagRec()` with that handle, or when another DB2 UDB CLI function call is made with that handle. However, information associated with a given handle type is not cleared by a call to `SQLGetDiagRec()` with an associated but different handle type. For example, a call to `SQLGetDiagRec()` with a connection handle input does not clear errors associated with any statement handles under that connection.

`SQL_SUCCESS` is returned even if the buffer for the error message (`szErrorMsg`) is too short, because the application is not able to retrieve the same error message by calling `SQLGetDiagRec()` again. The actual length of the message text is returned in the `pcbErrorMsg`.

To avoid truncation of the error message, declare a buffer length of `SQL_MAX_MESSAGE_LENGTH + 1`. The message text is never be longer than this.

### Return codes

- `SQL_SUCCESS`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`
- `SQL_NO_DATA_FOUND`

`SQL_NO_DATA_FOUND` is returned if no diagnostic information is available for the input handle, or if all of the messages have been retrieved through calls to `SQLGetDiagRec()`.

`SQL_ERROR` is returned if the argument `szSqlState`, `pfNativeError`, `szErrorMsg`, or `pcbErrorMsg` is a null pointer.

### Diagnostics

`SQLSTATEs` are not defined because `SQLGetDiagRec()` does not generate diagnostic information for itself.

### Restrictions

Although ODBC also returns X/Open SQL CAE `SQLSTATEs`, only DB2 UDB CLI returns the additional IBM defined `SQLSTATEs`. The ODBC Driver Manager also returns `SQLSTATE` values in addition to the standard ones. For more information about ODBC specific `SQLSTATEs` refer to *Microsoft ODBC Programmer's Reference*.

Because of this, you should only build dependencies on the standard `SQLSTATEs`. This means any branching logic in the application should only rely on the standard `SQLSTATEs`. The augmented `SQLSTATEs` are most useful for debugging purposes.

### References

- “`SQLGetDiagField` - Return diagnostic information (extensible)” on page 119

## SQLGetEnvAttr - Return current setting of an environment Attribute

### Purpose

`SQLGetEnvAttr()` returns the current settings for the specified environment attribute.

These options are set using the `SQLSetEnvAttr()` function.

## Syntax

```
SQLRETURN SQLGetEnvAttr (SQLHENV      henv,  
                        SQLINTEGER    Attribute,  
                        SQLPOINTER    Value,  
                        SQLINTEGER    BufferLength,  
                        SQLINTEGER    *StringLength);
```

## Function arguments

Table 90. SQLGetEnvAttr arguments

Data type	Argument	Use	Description
SQLHENV	<i>henv</i>	Input	Environment handle.
SQLINTEGER *	<i>StringLength</i>	Output	Length in bytes of the output data if the attribute value is a character string; otherwise, unused.
SQLINTEGER	<i>Attribute</i>	Input	Attribute to retrieve. Refer to Table 158 on page 197 for more information.
SQLINTEGER	<i>BufferLength</i>	Input	Maximum size of buffer pointed to by <i>Value</i> , if the attribute value is a character string; otherwise, unused.
SQLPOINTER	<i>Value</i>	Output	Current value associated with <i>Attribute</i> . The type of the value returned depends on <i>Attribute</i> .

If *Attribute* does not denote a string, then DB2 UDB CLI ignores *BufferLength* and does not set *StringLength*.

## 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.

## Diagnostics

Table 91. SQLGetEnvAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Attribute out of range	An <i>Attribute</i> value that is not valid is specified.  The argument <i>Value</i> or <i>StringLength</i> is a null pointer.

## SQLGetFunctions - Get functions

### Purpose

SQLGetFunctions() queries whether a specific function is supported. This allows applications to adapt to varying levels of support when using different drivers.

SQLConnect() must be called, and a connection to the data source (database server) must exist before calling this function.

## Syntax

```
SQLRETURN SQLGetFunctions (SQLHDBC          hdbc,  
                          SQLSMALLINT      fFunction,  
                          SQLSMALLINT      *pfSupported);
```

## Function arguments

Table 92. *SQLGetFunctions* arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Database connection handle.
SQLSMALLINT *	<i>pfSupported</i>	Output	Pointer to location where this function returns SQL_TRUE or SQL_FALSE depending on whether the function being queried is supported.
SQLSMALLINT	<i>fFunction</i>	Input	Function being queried.

## Usage

The following list shows the valid for the *fFunction* argument and whether the corresponding function is supported. Note that the values marked with an asterisk are not supported when connected to a remote server.

```
SQL_API_ALLOCCONNECT      = TRUE  
SQL_API_ALLOCENV         = TRUE  
SQL_API_ALLOCHANDLE     = TRUE  
SQL_API_ALLOCSTMT       = TRUE  
SQL_API_BINDCOL         = TRUE  
SQL_API_BINDFILETOCOL   = TRUE  
SQL_API_BINDFILETOPARAM = TRUE  
SQL_API_BINDPARAM       = TRUE  
SQL_API_BINDPARAMETER   = TRUE  
SQL_API_CANCEL          = TRUE  
SQL_API_CLOSECURSOR    = TRUE  
SQL_API_COLATTRIBUTES   = TRUE  
SQL_API_COLUMNS         = TRUE  
SQL_API_CONNECT         = TRUE  
SQL_API_COPYDESC        = TRUE  
SQL_API_DATASOURCES     = TRUE  
SQL_API_DESCRIBECOL     = TRUE  
SQL_API_DESCRIBEPARAM   = TRUE  
SQL_API_DISCONNECT      = TRUE  
SQL_API_DRIVERCONNECT   = TRUE  
SQL_API_ENDTRAN         = TRUE  
SQL_API_ERROR           = TRUE  
SQL_API_EXECDIRECT      = TRUE  
SQL_API_EXECUTE         = TRUE  
SQL_API_EXTENDEDFETCH   = TRUE  
SQL_API_FETCH           = TRUE  
SQL_API_FOREIGNKEYS     = TRUE  
SQL_API_FREECONNECT     = TRUE  
SQL_API_FREEENV         = TRUE  
SQL_API_FREEHANDLE      = TRUE  
SQL_API_FREESTMT        = TRUE  
SQL_API_GETCOL          = TRUE  
SQL_API_GETCONNECTATTR  = TRUE  
SQL_API_GETCONNECTOPTION = TRUE  
SQL_API_GETCURSORNAME   = TRUE  
SQL_API_GETDATA         = TRUE  
SQL_API_GETDESCFIELD    = TRUE  
SQL_API_GETDESCREC      = TRUE  
SQL_API_GETDIAGFIELD    = TRUE  
SQL_API_GETDIAGREC      = TRUE
```

SQL_API_GETENVATTR	= TRUE
SQL_API_GETFUNCTIONS	= TRUE
SQL_API_GETINFO	= TRUE
SQL_API_GETLENGTH	= TRUE
SQL_API_GETPOSITION	= TRUE
SQL_API_GETSTMTATTR	= TRUE
SQL_API_GETSTMTOPTION	= TRUE
SQL_API_GETSUBSTRING	= TRUE
SQL_API_GETTYPEINFO	= TRUE
SQL_API_LANGUAGES	= TRUE
SQL_API_MORERESULTS	= TRUE
SQL_API_NATIVESQL	= TRUE
SQL_API_NUMPARAMS	= TRUE
SQL_API_NUMRESULTCOLS	= TRUE
SQL_API_PARAMDATA	= TRUE
SQL_API_PARAMOPTIONS	= TRUE
SQL_API_PREPARE	= TRUE
SQL_API_PRIMARYKEYS	= TRUE
SQL_API_PROCEDURECOLUMNS	= TRUE
SQL_API_PROCEDURES	= TRUE
SQL_API_PUTDATA	= TRUE
SQL_API_RELEASEENV	= TRUE
SQL_API_ROWCOUNT	= TRUE
SQL_API_SETCONNECTATTR	= TRUE
SQL_API_COLATTRIBUTES	= TRUE
SQL_API_COLUMNS	= TRUE
SQL_API_CONNECT	= TRUE
SQL_API_COPYDESC	= TRUE
SQL_API_DATASOURCES	= TRUE
SQL_API_DESCRIBECOL	= TRUE
SQL_API_DESCRIBEPARAM	= TRUE
SQL_API_DISCONNECT	= TRUE
SQL_API_DRIVERCONNECT	= TRUE
SQL_API_ENDTRAN	= TRUE
SQL_API_ERROR	= TRUE
SQL_API_EXECDIRECT	= TRUE
SQL_API_EXECUTE	= TRUE
SQL_API_EXTENDEDFETCH	= TRUE
SQL_API_FETCH	= TRUE
SQL_API_FOREIGNKEYS	= TRUE
SQL_API_FREECONNECT	= TRUE
SQL_API_FREEENV	= TRUE
SQL_API_FREEHANDLE	= TRUE
SQL_API_FREESTMT	= TRUE
SQL_API_GETCOL	= TRUE
SQL_API_GETCONNECTATTR	= TRUE
SQL_API_GETCONNECTOPTION	= TRUE
SQL_API_GETCURSORNAME	= TRUE
SQL_API_GETDATA	= TRUE
SQL_API_GETDESCFIELD	= TRUE
SQL_API_GETDESCREC	= TRUE
SQL_API_GETDIAGFIELD	= TRUE
SQL_API_GETDIAGREC	= TRUE
SQL_API_GETENVATTR	= TRUE
SQL_API_GETFUNCTIONS	= TRUE
SQL_API_GETINFO	= TRUE
SQL_API_GETLENGTH	= TRUE
SQL_API_GETPOSITION	= TRUE
SQL_API_GETSTMTATTR	= TRUE
SQL_API_GETSTMTOPTION	= TRUE
SQL_API_GETSUBSTRING	= TRUE
SQL_API_GETTYPEINFO	= TRUE
SQL_API_LANGUAGES	= TRUE
SQL_API_MORERESULTS	= TRUE
SQL_API_NATIVESQL	= TRUE
SQL_API_NUMPARAMS	= TRUE
SQL_API_NUMRESULTCOLS	= TRUE

```

SQL_API_PARAMDATA      = TRUE
SQL_API_PARAMOPTIONS  = TRUE
SQL_API_PREPARE       = TRUE
SQL_API_PRIMARYKEYS   = TRUE
SQL_API_PROCEDURECOLUMNS = TRUE
SQL_API_PROCEDURES    = TRUE
SQL_API_PUTDATA       = TRUE
SQL_API_RELEASEENV    = TRUE
SQL_API_ROWCOUNT     = TRUE
SQL_API_SETCONNECTATTR = TRUE
SQL_API_SETCONNECTOPTION = TRUE
SQL_API_SETCURSORNAME = TRUE
SQL_API_SETDESCFIELD  = TRUE
SQL_API_SETDESCREC    = TRUE
SQL_API_SETENVATTR    = TRUE
SQL_API_SETPARAM      = TRUE
SQL_API_SETSTMTATTR   = TRUE
SQL_API_SETSTMTOPTION = TRUE
SQL_API_SPECIALCOLUMNS = TRUE *
SQL_API_STATISTICS    = TRUE *
SQL_API_TABLES        = TRUE
SQL_API_TRANSACT      = TRUE

```

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 93. SQLGetFunctions SQLSTATES

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid.	The argument <i>pfSupported</i> is a null pointer.
HY010	Function sequence error. Connection handles must not be allocated yet.	SQLGetFunctions is called before SQLConnect.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## SQLGetInfo - Get general information

### Purpose

SQLGetInfo() returns general information (including supported data conversions) about the Database Management System (DBMS) that the application is currently connected to.

## Syntax

```
SQLRETURN SQLGetInfo (SQLHDBC          hdbc,
                     SQLSMALLINT      fInfoType,
                     SQLPOINTER       rgbInfoValue,
                     SQLSMALLINT      cbInfoValueMax,
                     SQLSMALLINT      *pcbInfoValue);
```

## Function arguments

Table 94. SQLGetInfo arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Database connection handle.
SQLSMALLINT	<i>fInfoType</i>	Input	Type of the required information.
SQLPOINTER	<i>rgbInfoValue</i>	Output (also input)	Pointer to buffer where this function stores the required information. Depending on the type of information being retrieved, four types of information can be returned: <ul style="list-style-type: none"> <li>• 16-bit integer value</li> <li>• 32-bit integer value</li> <li>• 32-bit binary value</li> <li>• Null-terminated character string</li> </ul>
SQLSMALLINT	<i>cbInfoValueMax</i>	Input	The maximum length of the buffer pointed by <i>rgbInfoValue</i> pointer.
SQLSMALLINT *	<i>pcbInfoValue</i>	Output	Pointer to location where this function returns the total number of bytes available to return the required information.  If the value in the location pointed to by <i>pcbInfoValue</i> is greater than the size of the <i>rgbInfoValue</i> buffer as specified in <i>cbInfoValueMax</i> , then the string output information is truncated to <i>cbInfoValueMax</i> - 1 bytes and the function returns with SQL_SUCCESS_WITH_INFO.

## Usage

Table 95 lists the possible values of *fInfoType* and a description of the information that SQLGetInfo() returns for that value.

Table 95. Information returned by SQLGetInfo

<i>fInfoType</i>	Format	Description and notes
SQL_ACTIVE_CONNECTIONS	Short int	The maximum number of active connections supported per application.  Zero is returned, indicating that the limit is dependent on system resources.
SQL_ACTIVE_STATEMENTS	Short int	The maximum number of active statements per connection.  Zero is returned, indicating that the limit is dependent on system resources.

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	<b>Format</b>	<b>Description and notes</b>
SQL_AGGREGATE_FUNCTIONS	32-bit mask	A bit mask enumerating support for aggregation functions: <ul style="list-style-type: none"> <li>• SQL_AF_ALL</li> <li>• SQL_AF_AVG</li> <li>• SQL_AF_COUNT</li> <li>• SQL_AF_DISTINCT</li> <li>• SQL_AF_MAX</li> <li>• SQL_AF_MIN</li> <li>• SQL_AF_SUM</li> </ul>
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_COLUMN_ALIAS	String	Whether the connection supports column aliases. The value Y is returned if the connection supports the concept of a column alias.
SQL_CONNECTION_JOB_NAME	String	When in server mode, this is a character string that contains the complete job name associated with the connection. When not in server mode, a function sequence error is returned.

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	<b>Format</b>	<b>Description and notes</b>
SQL_CONVERT_BIGINT SQL_CONVERT_BINARY SQL_CONVERT_BLOB SQL_CONVERT_CHAR SQL_CONVERT_CLOB SQL_CONVERT_DATE SQL_CONVERT_DBCLOB SQL_CONVERT_DECIMAL SQL_CONVERT_DOUBLE SQL_CONVERT_FLOAT SQL_CONVERT_INTEGER SQL_CONVERT_LONGVARBINARY SQL_CONVERT_LONGVARCHAR SQL_CONVERT_NUMERIC SQL_CONVERT_REAL SQL_CONVERT_SMALLINT SQL_CONVERT_TIME SQL_CONVERT_TIMESTAMP SQL_CONVERT_VARBINARY SQL_CONVERT_VARCHAR SQL_CONVERT_WCHAR SQL_CONVERT_WLONGVARCHAR SQL_CONVERT_WVARCHAR	32-bit mask	<p>Indicates the conversions supported by the data source with the CONVERT scalar function for data of the type named in the infoType. If the bit mask equals zero, the data source does not support any conversions for the data of the named type, including conversions to the same data type.</p> <p>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 fInfoType of SQL_CONVERT_INTEGER. The application then ANDs the returned bit mask with SQL_CVT_DECIMAL. If the resulting value is nonzero, then the conversion is supported. The following bit masks are used to determine which conversions are supported:</p> <ul style="list-style-type: none"> <li>• SQL_CONVERT_BLOB</li> <li>• SQL_CONVERT_CLOB</li> <li>• SQL_CONVERT_DBCLOB</li> <li>• SQL_CONVERT_SMALLINT</li> <li>• SQL_CONVERT_TIME</li> <li>• SQL_CONVERT_TIMESTAMP</li> <li>• SQL_CONVERT_VARBINARY</li> <li>• SQL_CONVERT_VARCHAR</li> <li>• SQL_CONVERT_WCHAR</li> <li>• SQL_CONVERT_WLONGVARCHAR</li> <li>• SQL_CONVERT_WVARCHAR</li> <li>• SQL_CVT_BIGINT</li> <li>• SQL_CVT_BINARY</li> <li>• SQL_CVT_CHAR</li> <li>• SQL_CVT_DATE</li> <li>• SQL_CVT_DECIMAL</li> <li>• SQL_CVT_DOUBLE</li> <li>• SQL_CVT_FLOAT</li> <li>• SQL_CVT_INTEGER</li> <li>• SQL_CVT_LONGVARBINARY</li> <li>• SQL_CVT_LONGVARCHAR</li> <li>• SQL_CVT_NUMERIC</li> <li>• SQL_CVT_REAL</li> </ul>
SQL_CONVERT_FUNCTIONS	32 bit mask	<p>Indicates the scalar conversion functions supported by the driver and associated data source:</p> <ul style="list-style-type: none"> <li>• SQL_FN_CVT_CONVERT is used to determine which conversion functions are supported.</li> <li>• SQL_FN_CVT_CAST is used to determine which cast functions are supported.</li> </ul>



Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	Format	Description and notes
SQL_CORRELATION_NAME	Short int	Indicates the degree of correlation name support by the server: <ul style="list-style-type: none"> <li>• SQL_CN_ANY – Correlation name is supported and can be any valid user-defined name.</li> <li>• SQL_CN_NONE – Correlation name is not supported.</li> <li>• SQL_CN_DIFFERENT – Correlation name is supported but it must be different from the name of the table that it represents.</li> </ul>
SQL_CURSOR_COMMIT_BEHAVIOR	16-bit integer	Indicates how a COMMIT operation affects cursors: <ul style="list-style-type: none"> <li>• SQL_CB_DELETE destroys cursors and drops access plans for dynamic SQL statements.</li> <li>• SQL_CB_CLOSE destroys cursors, but retains access plans for dynamic SQL statements (including nonquery statements).</li> <li>• SQL_CB_PRESERVE retains cursors and access plans for dynamic statements (including nonquery statements). Applications can continue to fetch data, or close the cursor and reprocess the query without preparing the statement again.</li> </ul> <p><b>Note:</b> After the COMMIT operation, a FETCH must be issued to reposition the cursor before actions such as positioned updates or deletes can be taken.</p>
SQL_CURSOR_ROLLBACK_BEHAVIOR	16-bit integer	Indicates how a ROLLBACK operation affects cursors: <ul style="list-style-type: none"> <li>• SQL_CB_DELETE destroys cursors and drops access plans for dynamic SQL statements.</li> <li>• SQL_CB_CLOSE destroys cursors, but retains access plans for dynamic SQL statements (including nonquery statements)</li> <li>• SQL_CB_PRESERVE retains cursors and access plans for dynamic statements (including nonquery statements). Applications can continue to fetch data, or close the cursor and run the query again without preparing the statement again.</li> </ul> <p><b>Note:</b> DB2 servers do not have the SQL_CB_PRESERVE property.</p>
SQL_DATA_SOURCE_NAME	String	Name of the connected data source for the connection handle.
SQL_DATA_SOURCE_READ_ONLY	String	A character string of Y indicates that the database is set to READ ONLY mode; an N indicates that it is not set to READ ONLY mode.
SQL_DATABASE_NAME	String	Name of the current database in use. This string is the same as that returned by the SELECT CURRENT SERVER SQL statement.

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	<b>Format</b>	<b>Description and notes</b>
SQL_DBMS_NAME	String	Name of the Database Management System (DBMS) product being accessed.  For example: <ul style="list-style-type: none"> <li>• QSQ for DB2 Universal Database for iSeries</li> <li>• SQL for DB2 Universal Database for Linux, UNIX, and Windows</li> <li>• DSN for DB2 Universal Database for z/OS</li> </ul>
SQL_DBMS_VER	String	Version of the DBMS product accessed.
SQL_DEFAULT_TXN_ISOLATION	32-bit mask	The default transaction-isolation level supported.  One of the following masks are returned: <ul style="list-style-type: none"> <li>• SQL_TXN_READ_UNCOMMITTED – Changes are immediately perceived by all transactions (dirty read, non-repeatable read, and phantoms are possible). This is equivalent to UR level.</li> <li>• 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 CS level.</li> <li>• 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 RS level.</li> <li>• SQL_TXN_SERIALIZABLE – Data affected by pending transaction is not available to other transactions (repeatable read, phantoms are not possible). This is equivalent to RR level.</li> <li>• SQL_TXN_VERSIONING – Not applicable to IBM DBMSs.</li> <li>• SQL_TXN_NOCOMMIT – Any changes are effectively committed at the end of a successful operation; no explicit commit or rollback operation is allowed. This is a DB2 isolation level.</li> </ul> In IBM terminology, <ul style="list-style-type: none"> <li>• SQL_TXN_READ_UNCOMMITTED is uncommitted read.</li> <li>• SQL_TXN_READ_COMMITTED is cursor stability.</li> <li>• SQL_TXN_REPEATABLE_READ is read stability.</li> <li>• SQL_TXN_SERIALIZABLE is repeatable read.</li> </ul>
SQL_DESCRIBE_PARAMETER	String	Y if parameters can be described; N if not.
SQL_DRIVER_NAME	String	File name of the driver used to access the data source.

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	Format	Description and notes
SQL_DRIVER_ODBC_VER	String	The version number of ODBC that the driver supports. DB2 ODBC returns 2.1.
SQL_GROUP_BY	16-bit integer	Indicates the degree of support for the GROUP BY clause by the server: <ul style="list-style-type: none"> <li>• SQL_GB_NO_RELATION – There is no relationship between the columns in the GROUP BY and in the SELECT list.</li> <li>• SQL_GB_NOT_SUPPORTED – GROUP BY not supported.</li> <li>• SQL_GB_GROUP_BY_EQUALS_SELECT – GROUP BY must include all nonaggregated columns in the select list.</li> <li>• SQL_GB_GROUP_BY_CONTAINS_SELECT – GROUP BY clause must contain all nonaggregated columns in the SELECT list.</li> </ul>
SQL_IDENTIFIER_CASE	16-bit integer	Indicates case sensitivity of object names (such as table-name). <ul style="list-style-type: none"> <li>• SQL_IC_UPPER – Identifier names are stored in uppercase in the system catalog.</li> <li>• SQL_IC_LOWER – Identifier names are stored in lowercase in the system catalog.</li> <li>• SQL_IC_SENSITIVE – Identifier names are case sensitive, and are stored in mixed case in the system catalog.</li> <li>• SQL_IC_MIXED – Identifier names are not case sensitive, and are stored in mixed case in the system catalog.</li> </ul> <p><b>Note:</b> Identifier names in IBM DBMSs are not case sensitive.</p>
SQL_IDENTIFIER_QUOTE_CHAR	String	Character used as the delimiter of a quoted string.
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 that indicates whether an escape character is supported for the metacharacters percent and underscore in a LIKE predicate.
SQL_MAX_CATALOG_NAME_LEN	16-bit integer	The maximum length of a catalog qualifier name; first part of a three-part table name (in bytes).
SQL_MAX_COLUMN_NAME_LEN	Short int	The maximum length of a column name.
SQL_MAX_COLUMNS_IN_GROUP_BY	Short int	The maximum number of columns in a GROUP BY clause.
SQL_MAX_COLUMNS_IN_INDEX	Short int	The maximum number of columns in an SQL index.
SQL_MAX_COLUMNS_IN_ORDER_BY	Short int	Maximum number of columns in an ORDER BY clause.

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	<b>Format</b>	<b>Description and notes</b>
SQL_MAX_COLUMNS_IN_SELECT	Short int	The maximum number of columns in a SELECT statement.
SQL_MAX_COLUMNS_IN_TABLE	Short int	The maximum number of columns in an SQL table.
SQL_MAX_CURSOR_NAME_LEN	Short int	The maximum length of a cursor name.
SQL_MAX_OWNER_NAME_LEN	Short int	The maximum length of an owner name.
SQL_MAX_ROW_SIZE	32-bit unsigned integer	The maximum length in bytes that the server supports in single row of a base table. It is zero if there is no limit.
SQL_MAX_SCHEMA_NAME_LEN	Int	The maximum length of a schema name.
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	Short int	The maximum length of a table name.
SQL_MAX_TABLES_IN_SELECT	Short int	The maximum number of tables in a SELECT statement.
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.
SQL_NON_NULLABLE_COLUMNS	16-bit integer	Indicates whether non-nullable columns are supported: <ul style="list-style-type: none"> <li>• SQL_NNC_NON_NULL – columns can be defined as NOT NULL.</li> <li>• SQL_NNC_NULL – columns cannot be defined as NOT NULL.</li> </ul>

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	Format	Description and notes
SQL_NUMERIC_FUNCTIONS	32-bit mask	<p>The scalar numeric functions supported.</p> <p>The following bit masks are used to determine which numeric functions are supported:</p> <ul style="list-style-type: none"> <li>• SQL_FN_NUM_ABS</li> <li>• SQL_FN_NUM_ACOS</li> <li>• SQL_FN_NUM_ASIN</li> <li>• SQL_FN_NUM_ATAN</li> <li>• SQL_FN_NUM_ATAN2</li> <li>• SQL_FN_NUM_CEILING</li> <li>• SQL_FN_NUM_COS</li> <li>• SQL_FN_NUM_COT</li> <li>• SQL_FN_NUM_DEGREES</li> <li>• SQL_FN_NUM_EXP</li> <li>• SQL_FN_NUM_FLOOR</li> <li>• SQL_FN_NUM_LOG</li> <li>• SQL_FN_NUM_LOG10</li> <li>• SQL_FN_NUM_MOD</li> <li>• SQL_FN_NUM_PI</li> <li>• SQL_FN_NUM_POWER</li> <li>• SQL_FN_NUM_RADIANS</li> <li>• SQL_FN_NUM_RAND</li> <li>• SQL_FN_NUM_ROUND</li> <li>• SQL_FN_NUM_SIGN</li> <li>• SQL_FN_NUM_SIN</li> <li>• SQL_FN_NUM_SQRT</li> <li>• SQL_FN_NUM_TAN</li> <li>• SQL_FN_NUM_TRUNCATE</li> </ul>
SQL_ODBC_API_CONFORMANCE	16-bit integer	<p>The level of ODBC conformance:</p> <ul style="list-style-type: none"> <li>• SQL_OAC_NONE</li> <li>• SQL_OAC_LEVEL1</li> <li>• SQL_OAC_LEVEL2</li> </ul>
SQL_ODBC_SQL_CONFORMANCE	16-bit integer	<p>A value of:</p> <ul style="list-style-type: none"> <li>• SQL_OSC_MINIMUM means minimum ODBC SQL grammar supported</li> <li>• SQL_OSC_CORE means core ODBC SQL grammar supported</li> <li>• SQL_OSC_EXTENDED means extended ODBC SQL grammar supported</li> </ul> <p>For the definition of the previous types of ODBC SQL grammar, see Microsoft ODBC 3.0 Software Development Kit and Programmer's Reference.</p>
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	<p>The character string:</p> <ul style="list-style-type: none"> <li>• Y indicates that outer joins are supported, and DB2 ODBC supports the ODBC outer join request syntax.</li> <li>• N indicates that it is not supported.</li> </ul>
SQL_OWNER_TERM or SQL_SCHEMA_TERM	String	The database vendor terminology for a schema (owner).

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	<b>Format</b>	<b>Description and notes</b>
SQL_OWNER_USAGE or SQL_SCHEMA_USAGE	32-bit mask	Indicates the type of SQL statements that have schema (owners) associated with them when these statements are processed. Schema qualifiers (owners) are as follows: <ul style="list-style-type: none"> <li>• SQL_OU_DML_STATEMENTS is supported in all DML statements.</li> <li>• SQL_OU_PROCEDURE_INVOCATION is supported in the procedure invocation statement.</li> <li>• SQL_OU_TABLE_DEFINITION is supported in all table definition statements.</li> <li>• SQL_OU_INDEX_DEFINITION is supported in all index definition statements.</li> <li>• SQL_OU_PRIVILEGE_DEFINITION is supported in all privilege definition statements (that is, grant and revoke statements).</li> </ul>
SQL_POSITIONED_STATEMENTS	32-bit mask	Indicates the degree of support for positioned UPDATE and positioned DELETE statements: <ul style="list-style-type: none"> <li>• SQL_PS_POSITIONED_DELETE</li> <li>• SQL_PS_POSITIONED_UPDATE</li> <li>• SQL_PS_SELECT_FOR_UPDATE, indicates whether the server requires the FOR UPDATE clause to be specified on a &lt;query expression&gt; in order for a column to be updated with the cursor.</li> </ul>
SQL_PROCEDURE_TERM	String	Data source name for a procedure.
SQL_PROCEDURES	String	Whether the current server supports SQL procedures. The value Y is returned if the connection supports SQL procedures.
SQL_QUALIFIER_LOCATION or SQL_CATALOG_LOCATION	16-bit integer	A 16-bit integer value indicated the position of the qualifier in a qualified table name. Zero indicates that qualified names are not supported.
SQL_QUALIFIER_NAME_SEPARATOR or SQL_CATALOG_NAME_SEPARATOR	String	The characters used as a separator between a catalog name and the qualified name element that follows it.
SQL_QUALIFIER_TERM or SQL_CATALOG_TERM	String	The database vendor terminology for a qualifier.  This is the name that the vendor uses for the high-order part of a 3-part name.  Because DB2 ODBC does not support 3-part names, a zero-length string is returned.  For non-ODBC applications, the SQL_CATALOG_TERM symbolic name should be used instead of SQL_QUALIFIER_NAME.
SQL_QUALIFIER_USAGE or SQL_CATALOG_USAGE	32-bit mask	This is similar to SQL_OWNER_USAGE except that this is used for catalog.

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	Format	Description and notes
SQL_QUOTED_IDENTIFIER_CASE	16-bit integer	<ul style="list-style-type: none"> <li>• SQL_IC_UPPER – Quoted identifiers in SQL are case insensitive and stored in uppercase in the system catalog.</li> <li>• SQL_IC_LOWER – Quoted identifiers in SQL are case insensitive and are stored in lowercase in the system catalog.</li> <li>• SQL_IC_SENSITIVE – Quoted identifiers (delimited identifiers) in SQL are case sensitive and are stored in mixed case in the system catalog.</li> <li>• SQL_IC_MIXED – Quoted identifiers in SQL are case insensitive and are stored in mixed case in the system catalog.</li> </ul> <p>This should be contrasted with the SQL_IDENTIFIER_CASE <i>fInfoType</i>, which is used to determine how (unquoted) identifiers are stored in the system catalog.</p>
SQL_SEARCH_PATTERN_ESCAPE	String	Used to specify what the driver supports as an escape character for catalog functions, such as SQLTables() and SQLColumns().
SQL_SQL92_PREDICATES	32-bit mask	<p>Indicates the predicates supported in a SELECT statement that SQL-92 defines.</p> <ul style="list-style-type: none"> <li>• SQL_SP_BETWEEN</li> <li>• SQL_SP_COMPARISON</li> <li>• SQL_SP_EXISTS</li> <li>• SQL_SP_IN</li> <li>• SQL_SP_ISNOTNULL</li> <li>• SQL_SP_ISNULL</li> <li>• SQL_SP_LIKE</li> <li>• SQL_SP_MATCH_FULL</li> <li>• SQL_SP_MATCH_PARTIAL</li> <li>• SQL_SP_MATCH_UNIQUE_FULL</li> <li>• SQL_SP_MATCH_UNIQUE_PARTIAL</li> <li>• SQL_SP_OVERLAPS</li> <li>• SQL_SP_QUANTIFIED_COMPARISON</li> <li>• SQL_SP_UNIQUE</li> </ul>
SQL_SQL92_VALUE_EXPRESSIONS	32-bit mask	<p>Indicates the value expressions supported that SQL-92 defines.</p> <ul style="list-style-type: none"> <li>• SQL_SVE_CASE</li> <li>• SQL_SVE_CAST</li> <li>• SQL_SVE_COALESCE</li> <li>• SQL_SVE_NULLIF</li> </ul>

Table 95. Information returned by SQLGetInfo (continued)

<i>fInfoType</i>	Format	Description and notes
SQL_STRING_FUNCTIONS	32-bit bit mask	<p>Indicates which string functions are supported.</p> <p>The following bit masks are used to determine which string functions are supported:</p> <ul style="list-style-type: none"> <li>• SQL_FN_STR_ASCII</li> <li>• SQL_FN_STR_CHAR</li> <li>• SQL_FN_STR_CONCAT</li> <li>• SQL_FN_STR_DIFFERENCE</li> <li>• SQL_FN_STR_INSERT</li> <li>• SQL_FN_STR_LCASE</li> <li>• SQL_FN_STR_LEFT</li> <li>• SQL_FN_STR_LENGTH</li> <li>• SQL_FN_STR_LOCATE</li> <li>• SQL_FN_STR_LOCATE_2</li> <li>• SQL_FN_STR_LTRIM</li> <li>• SQL_FN_STR_REPEAT</li> <li>• SQL_FN_STR_REPLACE</li> <li>• SQL_FN_STR_RIGHT</li> <li>• SQL_FN_STR_RTRIM</li> <li>• SQL_FN_STR_SOUNDEX</li> <li>• SQL_FN_STR_SPACE</li> <li>• SQL_FN_STR_SUBSTRING</li> <li>• SQL_FN_STR_UCASE</li> </ul> <p>If an application can call the LOCATE scalar function with the string1, string2, and start arguments, the SQL_FN_STR_LOCATE bit mask is returned. If an application can only call the LOCATE scalar function with the string1 and string2, the SQL_FN_STR_LOCATE_2 bit mask is returned. If the LOCATE scalar function is fully supported, both bit masks are returned.</p>
SQL_TIMEDATE_FUNCTIONS	32-bit mask	<p>Indicates which time and date functions are supported.</p> <p>The following bit masks are used to determine which date functions are supported:</p> <ul style="list-style-type: none"> <li>• SQL_FN_TD_CURDATE</li> <li>• SQL_FN_TD_CURTIME</li> <li>• SQL_FN_TD_DAYNAME</li> <li>• SQL_FN_TD_DAYOFMONTH</li> <li>• SQL_FN_TD_DAYOFWEEK</li> <li>• SQL_FN_TD_DAYOFYEAR</li> <li>• SQL_FN_TD_HOUR</li> <li>• SQL_FN_TD_JULIAN_DAY</li> <li>• SQL_FN_TD_MINUTE</li> <li>• SQL_FN_TD_MONTH</li> <li>• SQL_FN_TD_MONTHNAME</li> <li>• SQL_FN_TD_NOW</li> <li>• SQL_FN_TD_QUARTER</li> <li>• SQL_FN_TD_SECOND</li> <li>• SQL_FN_TD_SECONDS_SINCE_MIDNIGHT</li> <li>• SQL_FN_TD_TIMESTAMPADD</li> <li>• SQL_FN_TD_TIMESTAMPDIFF</li> <li>• SQL_FN_TD_WEEK</li> <li>• SQL_FN_TD_YEAR</li> </ul>



Table 95. Information returned by *SQLGetInfo* (continued)

<i>fInfoType</i>	Format	Description and notes
SQL_TXN_CAPABLE	Short int	Indicates whether transactions can contain DDL or DML or both: <ul style="list-style-type: none"> <li>• SQL_TC_NONE – Transactions are not supported.</li> <li>• SQL_TC_DML – Transactions can only contain DML statements (SELECT, INSERT, UPDATE, DELETE, and so on). DDL statements (CREATE TABLE, DROP INDEX, and so on) encountered in a transaction cause an error.</li> <li>• SQL_TC_DDL_COMMIT – Transactions can only contain DML statements. DDL statements encountered in a transaction cause the transaction to be committed.</li> <li>• SQL_TC_DDL_IGNORE – Transactions can only contain DML statements. DDL statements encountered in a transaction are ignored.</li> <li>• SQL_TC_ALL – Transactions can contain DDL and DML statements in any order.</li> </ul>
SQL_USER_NAME	String	User name used in a particular database.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 96. *SQLGetInfo* SQLSTATEs

SQLSTATE	Description	Explanation
01004	Data truncated	The requested information is returned as a null-terminated string and its length exceeded the length of the application buffer as specified in <i>cbInfoValueMax</i> . The argument <i>pcbInfoValue</i> contains the actual (not truncated) length of the requested information.
08003	Connection not open	The type of information requested in <i>fInfoType</i> requires an open connection. Only SQL_ODBC_VER does not require an open connection.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>rgbInfoValue</i> is a null pointer  An <i>fInfoType</i> that is not valid is specified.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

# SQLGetLength - Retrieve length of a string value

## Purpose

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,  
                        SQLSMALLINT LocatorCType,  
                        SQLINTEGER Locator,  
                        SQLINTEGER *StringLength,  
                        SQLINTEGER *IndicatorValue);
```

## Function arguments

Table 97. SQLGetLength arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	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.
SQLINTEGER *	<i>IndicatorValue</i>	Output	Always set to zero.
SQLINTEGER *	<i>StringLength</i>	Output	The length of the specified locator. <sup>1</sup>  If the pointer is set to NULL then the SQLSTATE HY009 is returned.
SQLINTEGER	<i>Locator</i>	Input	Must be set to the LOB locator value.
SQLSMALLINT	<i>LocatorCType</i>	Input	The C type of the source LOB locator. <ul style="list-style-type: none"><li>• SQL_C_BLOB_LOCATOR</li><li>• SQL_C_CLOB_LOCATOR</li><li>• SQL_C_DBCLOB_LOCATOR</li></ul>

1. This is in bytes even for DBCLOB data.

## Usage

SQLGetLength() can be used to determine the length of the data value represented by an 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 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 is created has terminated.

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

## Error conditions

Table 98. SQLGetLength SQLSTATES

SQLSTATE	Description	Explanation
07006	Conversion that is not valid	The combination of the argument <i>LocatorCType</i> and <i>Locator</i> is not valid.
0F001	LOB variable that is not valid	The value specified for the argument <i>Locator</i> has not been associated with an LOB locator.
58004	Unexpected system failure	Unrecoverable system error.
HY003	Program type out of range	The argument <i>LocatorCType</i> is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY009	Argument value that is not valid	The argument <i>StringLength</i> or <i>IndicatorValue</i> is a null pointer.
HY010	Function sequence error	The specified argument <i>StatementHandle</i> is not in an <i>allocated</i> state.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
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 a DB2 server that does not support Large Objects.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLFetch - Fetch next row” on page 87
- “SQLGetPosition - Return starting position of string”
- “SQLGetSubString - Retrieve portion of a string value” on page 146

## SQLGetPosition - Return starting position of string

### Purpose

SQLGetPosition() is used to return the starting position of one string within an LOB value (the source). The source value must be an LOB locator, the search string can be an 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,
                          SQLSMALLINT LocatorCType,
                          SQLINTEGER SourceLocator,
                          SQLINTEGER SearchLocator,
                          SQLCHAR *SearchLiteral,
                          SQLINTEGER SearchLiteralLength,
                          SQLINTEGER FromPosition,
                          SQLINTEGER *LocatedAt,
                          SQLINTEGER *IndicatorValue);
```

## Function arguments

Table 99. SQLGetPosition arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>SearchLiteral</i>	Input	This argument points to the area of storage that contains the search string literal.  If <i>SearchLiteralLength</i> is 0, this pointer must be NULL.
SQLHSTMT	<i>StatementHandle</i>	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.
SQLINTEGER *	<i>IndicatorValue</i>	Output	Always set to zero.
SQLINTEGER *	<i>LocatedAt</i>	Output	For BLOBs and CLOBs, this is the byte position at which the string is 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	<i>FromPosition</i>	Input	For BLOBs and CLOBs, this is the position of the first byte within the source string at which the search is to start. to be returned by the function. For DBCLOBs, this is the first character. The start byte or character is numbered 1.
SQLINTEGER	<i>SearchLiteralLength</i>	Input	The length of the string in <i>SearchLiteral</i> (in bytes). <sup>1</sup>  If this argument value is 0, then the argument <i>SearchLocator</i> is meaningful.
SQLINTEGER	<i>SearchLocator</i>	Input	If the <i>SearchLiteral</i> pointer is NULL and if <i>SearchLiteralLength</i> is set to 0, then <i>SearchLocator</i> must be set to the LOB locator associated with the search string; otherwise, this argument is ignored.
SQLINTEGER	<i>SourceLocator</i>	Input	<i>SourceLocator</i> must be set to the source LOB locator.
SQLSMALLINT	<i>LocatorCType</i>	Input	The C type of the source LOB locator. This can be: <ul style="list-style-type: none"> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> </ul>

1. This is in bytes even for DBCLOB data.

## Usage

SQLGetPosition() is used in conjunction with SQLGetSubString() in order to obtain any portion of a string 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 is created has terminated.

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

## Error conditions

Table 100. SQLGetPosition SQLSTATES

SQLSTATE	Description	Explanation
07006	Conversion that is not valid	The combination of the <i>LocatorCType</i> argument and either of the LOB locator values is not valid.
0F001	LOB variable that is not valid	The value specified for argument <i>Locator</i> or <i>SearchLocator</i> is not currently an LOB locator.
42818	Length that is not valid	The length of the pattern is too long.
58004	Unexpected system failure	Unrecoverable system error.
HY009	Argument value that is not valid	The argument <i>LocatedAt</i> or <i>IndicatorValue</i> is a null pointer.  The argument value for <i>FromPosition</i> is not greater than 0.  <i>LocatorCType</i> is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY010	Function sequence error	The specified <i>StatementHandle</i> argument is not in an <i>allocated</i> state.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value of <i>SearchLiteralLength</i> is less than 1, and not SQL_NTS.
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 a DB2 server that does not support Large Objects.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLExtendedFetch - Fetch array of rows” on page 85
- “SQLFetch - Fetch next row” on page 87
- “SQLGetLength - Retrieve length of a string value” on page 140
- “SQLGetSubString - Retrieve portion of a string value” on page 146

## SQLGetStmtAttr - Get the value of a statement attribute

### Purpose

SQLGetStmtAttr() returns the current settings of the specified statement attribute.

These options are set using the `SQLSetStmtAttr()` function. This function is similar to `SQLGetStmtOption()`, both functions are supported for compatibility reasons.

## Syntax

```
SQLRETURN SQLGetStmtAttr( SQLHSTMT      hstmt,
                          SQLINTEGER    fAttr,
                          SQLPOINTER    pvParam,
                          SQLINTEGER    bLen,
                          SQLINTEGER    *sLen);
```

## Function arguments

Table 101. `SQLGetStmtAttr` arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER	<i>fAttr</i>	Input	Attribute to retrieve. Refer to Table 102 for more information.
SQLPOINTER	<i>pvParam</i>	Output	Pointer to buffer for requested attribute.
SQLINTEGER	<i>bLen</i>	Input	Maximum number of bytes to store in <i>pvParam</i> , if the attribute is a character string; otherwise, unused.
SQLINTEGER *	<i>sLen</i>	Output	Length of output data if the attribute is a character string; otherwise, unused.

## Usage

Table 102. Statement attributes

<i>fAttr</i>	Data type	Contents
SQL_ATTR_APP_PARAM_DESC	Integer	The descriptor handle used by the application to provide parameter values for this statement handle.
SQL_ATTR_APP_ROW_DESC	Integer	The descriptor handle for the application to retrieve row data using the statement handle.
SQL_ATTR_CURSOR_SCROLLABLE	Integer	A 32-bit integer value that specifies if cursors opened for this statement handle should be scrollable. <ul style="list-style-type: none"> <li>SQL_FALSE – Cursors are not scrollable, and <code>SQLFetchScroll()</code> cannot be used against them. This is the default.</li> <li>SQL_TRUE – Cursors are scrollable. <code>SQLFetchScroll()</code> can be used to retrieve data from these cursors.</li> </ul>
SQL_ATTR_CURSOR_TYPE	Integer	A 32-bit integer value that specifies the behavior of cursors opened for this statement handle. <ul style="list-style-type: none"> <li>SQL_CURSOR_FORWARD_ONLY – Cursors are not scrollable, and <code>SQLFetchScroll()</code> cannot be used against them. This is the default.</li> <li>SQL_DYNAMIC – Cursors are scrollable. <code>SQLFetchScroll()</code> can be used to retrieve data from these cursors.</li> </ul>
SQL_ATTR_FOR_FETCH_ONLY	Integer	Indicates if cursors opened for this statement handle should be read-only. <ul style="list-style-type: none"> <li>SQL_FALSE – Cursors can be used for positioned updates and deletes. This is the default.</li> <li>SQL_TRUE – Cursors are read-only and cannot be used for positioned updates or deletes.</li> </ul>
SQL_ATTR_IMP_PARAM_DESC	Integer	The descriptor handle used by the CLI implementation to provide parameter values for this statement handle.
SQL_ATTR_IMP_ROW_DESC	Integer	The descriptor handle used by the CLI implementation to retrieve row data using this statement handle.
SQL_ATTR_ROWSET_SIZE	Integer	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 <code>SQLExtendedFetch()</code> . The default value is 1.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR

- SQL\_INVALID\_HANDLE

## Diagnostics

Table 103. SQLGetStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>pvParam</i> is a null pointer. An <i>fAttr</i> that is not valid value is specified.
HYC00	Driver not capable	DB2 UDB CLI recognizes the option but does not support it.

## SQLGetStmtOption - Return current setting of a statement option

### Purpose

SQLGetStmtOption() has been deprecated and replaced with SQLGetStmtAttr(). Although this version of DB2 UDB CLI continues to support SQLGetStmtOption(), it is recommended that you begin using SQLGetStmtAttr() in your DB2 UDB CLI programs so that they conform to the latest standards.

SQLGetStmtOption() returns the current settings of the specified statement option.

These options are set using the SQLSetStmtOption() function.

### Syntax

```
SQLRETURN SQLGetStmtOption( SQLHSTMT      hstmt,
                             SQLSMALLINT  fOption,
                             SQLPOINTER   pvParam);
```

### Function arguments

Table 104. SQLStmtOption arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Connection handle.
SQLPOINTER	<i>pvParam</i>	Output	Value of the option. Depending on the value of <i>fOption</i> this can be a 32-bit integer value, or a pointer to a null terminated character string.
SQLSMALLINT	<i>fOption</i>	Input	Option to retrieve. See Table 102 on page 144 for more information.

### Usage

SQLGetStmtOption() provides the same function as SQLGetStmtAttr(), both functions are supported for compatibility reasons.

See Table 102 on page 144 for a list of statement options.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 105. SQLStmtOption SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>pvParam</i> is a null pointer.  A <i>fOption</i> that is not valid value is specified.
HYC00	Driver not capable	DB2 UDB CLI recognizes the option but does not support it.

## SQLGetSubString - Retrieve portion of a string value

### Purpose

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,
    SQLSMALLINT       LocatorCType,
    SQLINTEGER         SourceLocator,
    SQLINTEGER         FromPosition,
    SQLINTEGER         ForLength,
    SQLSMALLINT       TargetCType,
    SQLPOINTER         DataPtr,
    SQLINTEGER         BufferLength,
    SQLINTEGER         *StringLength,
    SQLINTEGER         *IndicatorValue);
```

### Function arguments

Table 106. SQLGetSubString arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	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.
SQLINTEGER *	<i>IndicatorValue</i>	output	Always set to zero.
SQLINTEGER *	<i>StringLength</i>	output	The length of the returned information in <i>DataPtr</i> in bytes <sup>a</sup> 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	<i>BufferLength</i>	input	Maximum size of the buffer pointed to by <i>DataPtr</i> in bytes.



Table 106. *SQLGetSubString* arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>ForLength</i>	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 <i>FromPosition</i> is less than the length of the source string but <i>FromPosition</i> + <i>ForLength</i> - 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).
SQLINTEGER	<i>FromPosition</i>	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.
SQLINTEGER	<i>SourceLocator</i>	input	<i>SourceLocator</i> must be set to the source LOB locator value.
SQLPOINTER	<i>DataPtr</i>	output	Pointer to the buffer where the retrieved string value or an LOB locator is to be stored.
SQLSMALLINT	<i>LocatorCType</i>	input	The C type of the source LOB locator. This can be: <ul style="list-style-type: none"> <li>• SQL_C_BLOB_LOCATOR</li> <li>• SQL_C_CLOB_LOCATOR</li> <li>• SQL_C_DBCLOB_LOCATOR</li> </ul>
SQLSMALLINT	<i>TargetCType</i>	input	The C data type of the <i>DataPtr</i> . The target must be a C string variable (SQL_C_CHAR, SQL_C_WCHAR, SQL_C_BINARY, or SQL_C_DBCHAR).

**Note:** 1. 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 data in pieces. In this case a column is first bound to an 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 is created has terminated.

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

## Error conditions

Table 107. SQLGetSubString SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated	The amount of data to be returned is longer than <i>BufferLength</i> . Actual length available for return is stored in <i>StringLength</i> .
07006	Conversion that is not valid	The value specified for <i>TargetCType</i> is not SQL_C_CHAR, SQL_C_BINARY, SQL_C_DBCHAR, or an LOB locator.  The value specified for <i>TargetCType</i> is inappropriate for the source (for example SQL_C_DBCHAR for a BLOB column).
22011	Substring error occurred	<i>FromPosition</i> is greater than the length of the source string.
58004	Unexpected system failure	Unrecoverable system error.
HY003	Program type out of range	<i>LocatorCType</i> is not one of SQL_C_CLOB_LOCATOR, SQL_C_BLOB_LOCATOR, or SQL_C_DBCLOB_LOCATOR.
HY009	Argument value that is not valid	The value specified for <i>FromPosition</i> or <i>ForLength</i> is not a positive integer.  The argument <i>DataPtr</i> , <i>StringLength</i> , or <i>IndicatorValue</i> is a null pointer
HY010	Function sequence error	The specified <i>StatementHandle</i> is not in an <i>allocated</i> state.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value of <i>BufferLength</i> is 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 <i>Locator</i> is not currently an LOB locator.

## Restrictions

This function is not available when connected to a DB2 server that does not support Large Objects.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLFetch - Fetch next row” on page 87
- “SQLGetData - Get data from a column” on page 114
- “SQLGetLength - Retrieve length of a string value” on page 140
- “SQLGetPosition - Return starting position of string” on page 141

## SQLGetTypeInfo - Get data type information

### Purpose

SQLGetTypeInfo() returns information about the data types that are supported by the Database Management Systems (DBMSs) associated with DB2 UDB 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,  
                          SQLSMALLINT DataType);
```

### Function arguments

Table 108. SQLGetTypeInfo arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle
SQLSMALLINT	<i>DataType</i>	Input	The SQL data type being queried. The supported types are: <ul style="list-style-type: none"><li>• SQL_ALL_TYPES</li><li>• SQL_BIGINT</li><li>• SQL_BINARY</li><li>• SQL_BLOB</li><li>• SQL_CHAR</li><li>• SQL_CLOB</li><li>• SQL_DATE</li><li>• SQL_DBCLOB</li><li>• SQL_DECIMAL</li><li>• SQL_DOUBLE</li><li>• SQL_FLOAT</li><li>• SQL_GRAPHIC</li><li>• SQL_INTEGER</li><li>• SQL_NUMERIC</li><li>• SQL_REAL</li><li>• SQL_SMALLINT</li><li>• SQL_TIME</li><li>• SQL_TIMESTAMP</li><li>• SQL_VARBINARY</li><li>• SQL_VARCHAR</li><li>• SQL_VARGRAPHIC</li></ul> If SQL_ALL_TYPES is specified, information about all supported data types is returned in ascending order by TYPE_NAME. All unsupported data types are absent from the result set.

### Usage

Because SQLGetTypeInfo() generates a result set and is equivalent to executing a query, it generates a cursor and begins a transaction. To prepare and process another statement on this statement handle, the cursor must be closed.

If `SQLGetTypeInfo()` is called with a *Data Type* that is not valid, an empty result set is returned.

The columns of the result set that is generated by this function are described below.

Although new columns might be added and the names of the existing columns might be changed in future releases, the position of the current columns does not change. The data types that are returned are those that can be used in a `CREATE TABLE`, `ALTER TABLE`, `DDL` statement. Nonpersistent data types are not part of the returned result set. User-defined data types are not returned either.

Table 109. Columns returned by `SQLGetTypeInfo`

Column number/name	Data type	Description
1 TYPE_NAME	VARCHAR(128) NOT NULL	Character representation of the SQL data type name (for example, VARCHAR, DATE, INTEGER)
2 DATA_TYPE	SMALLINT NOT NULL	SQL data type define values (for example, SQL_VARCHAR, SQL_DATE, SQL_INTEGER)
3 COLUMN_SIZE	INTEGER	<p>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.</p> <p>For date, time, timestamp data types, this is the total number of characters required to display the value when converted to character.</p> <p>For numeric data types, this is the total number of digits.</p>
4 LITERAL_PREFIX	VARCHAR(128)	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.
5 LITERAL_SUFFIX	VARCHAR(128)	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.
6 CREATE_PARAMS	VARCHAR(128)	<p>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: LENGTH, PRECISION, SCALE. They appear in the order that the SQL syntax requires that they be used.</p> <p>A NULL indicator is returned if there are no parameters for the data type definition, (such as INTEGER).</p> <p><b>Note:</b> The intent of CREATE_PARAMS is to enable an application to customize the interface for a <i>DDL builder</i>. 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 can be used to label an edit control.</p>
7 NULLABLE	SMALLINT NOT NULL	<p>Indicates whether the data type accepts a NULL value</p> <ul style="list-style-type: none"> <li>• Set to SQL_NO_NULLS if NULL values are disallowed.</li> <li>• Set to SQL_NULLABLE if NULL values are allowed.</li> <li>• Set to SQL_NULLABLE_UNKNOWN if it is not known whether NULL values are allowed or not.</li> </ul>

Table 109. Columns returned by SQLGetTypeInfo (continued)

Column number/name	Data type	Description
8 CASE_SENSITIVE	SMALLINT NOT NULL	Indicates whether the data type can be treated as case sensitive for collation purposes; valid values are SQL_TRUE and SQL_FALSE.
9 SEARCHABLE	SMALLINT NOT NULL	Indicates how the data type is used in a WHERE clause. Valid values are: <ul style="list-style-type: none"> <li>SQL_UNSEARCHABLE – if the data type cannot be used in a WHERE clause.</li> <li>SQL_LIKE_ONLY – if the data type can be used in a WHERE clause only with the LIKE predicate.</li> <li>SQL_ALL_EXCEPT_LIKE – if the data type can be used in a WHERE clause with all comparison operators except LIKE.</li> <li>SQL_SEARCHABLE – if the data type can be used in a WHERE clause with any comparison operator.</li> </ul>
10 UNSIGNED_ATTRIBUTE	SMALLINT	Indicates where 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.
11 FIXED_PREC_SCALE	SMALLINT NOT NULL	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.
12 AUTO_INCREMENT	SMALLINT	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.
13 LOCAL_TYPE_NAME	VARCHAR(128)	This column contains any localized 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.
14 MINIMUM_SCALE	INTEGER	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.
15 MAXIMUM_SCALE	INTEGER	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.
16 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 interval and datetime data types which DB2 CLI does not support).
17 SQL_DATETIME_SUB	SMALLINT	This field is always NULL (DB2 CLI does not support interval and datetime data types).

Table 109. Columns returned by *SQLGetTypeInfo* (continued)

Column number/name	Data type	Description
18 NUM_PREC_RADIX	INTEGER	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.
19 INTERVAL_PRECISION	SMALLINT	This field is always NULL (DB2 CLI does not support interval data types).

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Error conditions

Table 110. *SQLGetTypeInfo* SQLSTATES

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	A cursor is already opened on the statement handle. <i>StatementHandle</i> has not been closed.
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY004	SQL data type out of range	A <i>DataType</i> that is not valid is specified.
HY010	Function sequence error	The function is called while in a data-at-processing (SQLParamData(), SQLPutData()) operation.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYT00	Timeout expired	

## Restrictions

The following ODBC specified SQL data types (and their corresponding *DataType* define values) are not supported by any IBM RDBMS:

Data type	<i>DataType</i>
TINY INT	SQL_TINYINT
BIT	SQL_BIT

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```
/* From CLI sample typeinfo.c */
/* ... */
rc = SQLGetTypeInfo(hstmt, SQL_ALL_TYPES);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );
```

```

rc = SQLBindCol(hstmt, 1, SQL_C_CHAR, (SQLPOINTER) typename.s, 128, &typename.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 2, SQL_C_DEFAULT, (SQLPOINTER) &datatype,
               sizeof(datatype), &datatype_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 3, SQL_C_DEFAULT, (SQLPOINTER) &precision,
               sizeof(precision), &precision_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 7, SQL_C_DEFAULT, (SQLPOINTER) &nullable,
               sizeof(nullable), &nullable_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 8, SQL_C_DEFAULT, (SQLPOINTER) &casesens,
               sizeof(casesens), &casesens_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

printf("Datatype          Datatype Precision Nullable Case\n");
printf("Typename          (int)                Sensitive\n");
printf("-----\n");
/* LONG VARCHAR FOR BIT DATA      99 2147483647  FALSE  FALSE */
/* Fetch each row, and display */
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    printf("%-25s ", typename.s);
    printf("%8d ", datatype);
    printf("%10ld ", precision);
    printf("%-8s ", truefalse[nullable]);
    printf("%-9s\n", truefalse[casesens]);
}
/* endwhile */

if ( rc != SQL_NO_DATA_FOUND )
    CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

```

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLGetInfo - Get general information” on page 127

## SQLLanguages - Get SQL dialect or conformance information

### Purpose

SQLLanguages() returns SQL dialect or conformance information. 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 SELECT statement.

### Syntax

```
SQLRETURN SQLLanguages (SQLHSTMT hstmt);
```

### Function arguments

Table 111. SQLLanguages arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle

## Usage

The function returns dialect and conformance information, in the form of a result set on `StatementHandle`. This contains a row for every conformance claim the SQL product makes (including subsets defined for ISO and vendor-specific versions). For a product that claims to comply with this specification, the result set thus contains at least one row.

Rows defining ISO standard and vendor-specific languages can exist in the same table. Each row has at least these columns and, if it makes an X/Open SQL conformance claim, the columns contains these values.

Table 112. Columns returned by `SQLLanguages`

Column name	Data type	Description
BINDING_SYTLE	VARCHAR(254)	Either 'EMBEDDED', 'DIRECT', OR 'CLI'.
CONFORMANCE	VARCHAR(254)	The conformance level to the relevant document that the implementation claims.
IMPLEMENTATION	VARCHAR(254)	A character string, defined by the vendor, that uniquely identifies the vendor's SQL product.
INTEGRITY	VARCHAR(254)	An indication of whether the implementation supports the Integrity Enhancement Feature (IEF).
PROGRAMMING_LANG	VARCHAR(254)	The host language for which the binding style is supported.
SOURCE_YEAR	VARCHAR(254)	The year the relevant source document is approved.
SOURCE	VARCHAR(254), NOT NULL	The organization that defined this SQL version.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 113. `SQLLanguages SQLSTATES`

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	Cursor related information is requested, but no cursor is open.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal SQL_NTS.
HYC00	Driver not capable	DB2 UDB CLI does not support <i>catalog</i> as a qualifier for table name.



## SQLMoreResults - Determine whether there are more result sets

### Purpose

SQLMoreResults() determines whether there is more information available on the statement handle which has been associated with a stored procedure that is returning result sets.

### Syntax

```
SQLRETURN SQLMoreResults (SQLHSTMT StatementHandle);
```

### Function arguments

Table 114. SQLMoreResults arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	input	Statement handle

### Usage

This function is used to return multiple results that are set in a sequential manner upon the processing of a stored procedure that contains SQL queries. The cursors have been left open so that the result sets remain accessible when the stored procedure has finished processing.

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.

If SQLFreeStmt() is called with the SQL\_CLOSE or SQL\_DROP option, 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

### Error conditions

Table 115. SQLMoreResults SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
58004	Unexpected system failure	Unrecoverable system error.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	The function is called while in a data-at-processing (SQLParamData(), SQLPutData()) operation.

Table 115. *SQLMoreResults SQLSTATES (continued)*

SQLSTATE	Description	Explanation
HY013	Unexpected memory handling error	DB2 UDB CLI is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYT00	Timeout expired	

In addition `SQLMoreResults()` can return the `SQLSTATES` associated with `SQLExecute()`.

## Restrictions

The ODBC specification of `SQLMoreResults()` also allow counts associated with the processing of parameterized INSERT, UPDATE, and DELETE statements with arrays of input parameter values to be returned. However, DB2 UDB CLI does not support the return of such count information.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLBindParameter - Bind a parameter marker to a buffer” on page 42

## SQLNativeSql - Get native SQL text

### Purpose

`SQLNativeSql()` is used to show how DB2 UDB CLI interprets vendor escape clauses. If the original SQL string passed in by the application contains vendor escape clause sequences, then DB2 UDB CLI returns the transformed SQL string that is seen by the data source (with vendor escape clauses either converted or discarded, as appropriate).

### Syntax

```
SQLRETURN SQLNativeSql (SQLHDBC          ConnectionHandle,
                        SQLCHAR          *InStatementText,
                        SQLINTEGER       TextLength1,
                        SQLCHAR          *OutStatementText,
                        SQLINTEGER       BufferLength,
                        SQLINTEGER       *TextLength2Ptr);
```

## Function arguments

Table 116. *SQLNativeSql arguments*

Data type	Argument	Use	Description
SQLCHAR *	<i>InStatementText</i>	Input	Input SQL string.
SQLCHAR *	<i>OutStatementText</i>	Output	Pointer to buffer for the transformed output string.
SQLHDBC	<i>ConnectionHandle</i>	Input	Connection handle.
SQLINTEGER *	<i>TextLength2Ptr</i>	Output	The total number of bytes available to return in <i>OutStatementText</i> . If the number of bytes available to return is greater than or equal to <i>BufferLength</i> , the output SQL string in <i>OutStatementText</i> is truncated to <i>BufferLength - 1</i> bytes. The value <code>SQL_NULL_DATA</code> is returned if no output string is generated.
SQLINTEGER	<i>BufferLength</i>	Input	Size of buffer pointed by <i>OutStatementText</i> .

Table 116. *SQLNativeSql* arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>TextLength1</i>	Input	Length of <i>InStatementText</i> .

## Usage

This function is called when the application wants to examine or display the transformed SQL string that is passed to the data source by DB2 UDB CLI. Translation (mapping) only occurs if the input SQL statement string contains vendor escape clause sequences.

There are no vendor escape sequences on iSeries; this procedure is provided for compatibility purposes. Also, note that this procedure can be used to evaluate an SQL string for syntax errors.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Error conditions

Table 117. *SQLNativeSql* SQLSTATES

SQLSTATE	Description	Explanation
01004	Data truncated	The buffer <i>OutStatementText</i> is not large enough to contain the entire SQL string, so truncation occurred. The argument <i>TextLength2Ptr</i> contains the total length of the untruncated SQL string. (Function returns with SQL_SUCCESS_WITH_INFO.)
08003	Connection is closed	The <i>ConnectionHandle</i> does not reference an open database connection.
37000	SQL syntax that is not valid	The input SQL string in <i>InStatementText</i> contained a syntax error.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>InStatementText</i> , <i>OutStatementText</i> , or <i>TextLength2Ptr</i> is a null pointer.
HY090	String or buffer length that is not valid	The argument <i>TextLength1</i> is less than 0, but not equal to SQL_NTS.  The argument <i>BufferLength</i> is less than 0.

## Restrictions

None.

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/* From CLI sample native.c */
/* ... */
SQLCHAR in_stmt[1024], out_stmt[1024] ;
SQLSMALLINT pcPar ;
SQLINTEGER indicator ;

```

```

/* ... */
/* Prompt for a statement to prepare */
printf("Enter an SQL statement: \n");
gets((char *)in_stmt);

/* prepare the statement */
rc = SQLPrepare(hstmt, in_stmt, SQL_NTS);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

SQLNumParams(hstmt, &pcPar);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

SQLNativeSql(hstmt, in_stmt, SQL_NTS, out_stmt, 1024, &indicator);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

if ( indicator == SQL_NULL_DATA ) printf( "Invalid statement\n" ) ;
else {
    printf( "Input Statement: \n %s \n", in_stmt ) ;
    printf( "Output Statement: \n %s \n", in_stmt ) ;
    printf( "Number of Parameter Markers = %d\n", pcPar ) ;
}

rc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc ) ;

```

## References

None.

## SQLNextResult - Process the next result set

### Purpose

SQLNextResult() determines whether there is more information available on the statement handle that has been associated with a stored procedure that is returning result sets.

### Syntax

```
SQLRETURN SQLNextResult (SQLHSTMT StatementHandle,
                        SQLHSTMT NextResultHandle);
```

### Function arguments

Table 118. SQLNextResult arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLHSTMT	<i>NextResultHandle</i>	Input	Statement handle for next result set.

### Usage

This function is used to associate the next result set from StatementHandle with NextResultHandle. This differs from SQLMoreResults() because it allows both statement handles to process their result sets simultaneously.

If all the result sets have been processed, SQLNextResult() returns SQL\_NO\_DATA\_FOUND.

If SQLFreeStmt() is called with the SQL\_CLOSE or SQL\_DROP option, all pending result sets on this statement handle are discarded.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NO\_DATA\_FOUND

## Error conditions

Table 119. SQLNextResult SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
58004	Unexpected system failure	Unrecoverable system error.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	The function is called while in a data-at-processing (SQLParamData(), SQLPutData()) operation.
HY013	Unexpected memory handling error	DB2 UDB CLI is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYT00	Timeout expired	

## References

- “SQLMoreResults - Determine whether there are more result sets” on page 155

## SQLNumParams - Get number of parameters in an SQL statement

### Purpose

SQLNumParams() returns the number of parameter markers in an SQL statement.

### Syntax

```
SQLRETURN SQLNumParams (SQLHSTMT StatementHandle,  
SQLSMALLINT *ParameterCountPtr);
```

### Function arguments

Table 120. SQLNumParams arguments

Data type	Argument	Use	Description
SQLHSTMT	StatementHandle	Input	Statement handle.
SQLSMALLINT *	ParameterCountPtr	Output	Number of parameters in the statement.

### Usage

This function can only be called after the statement that is 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() 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

## Error conditions

Table 121. SQLNumParams SQLSTATES

SQLSTATE	Description	Explanation
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY008	Operation canceled	
HY009	Argument value that is not valid	<i>ParameterCountPtr</i> is null.
HY010	Function sequence error	This function is called before SQLPrepare() is called for the specified <i>StatementHandle</i>  The function is called while in a data-at-processing (SQLParamData(), SQLPutData()) operation.
HY013	Unexpected memory handling error	DB2 UDB CLI is unable to access memory required to support the processing or completion of the function.
HYT00	Timeout expired	

## Restrictions

None.

## Example

Refer to the SQLNativeSql() "" on page 157.

## References

- "SQLBindParam - Bind a buffer to a parameter marker" on page 38
- "SQLPrepare - Prepare a statement" on page 164

## SQLNumResultCols - Get number of result columns

### Purpose

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 SQLDescribeCol(), SQLColAttributes(), SQLBindCol() or SQLGetData().

## Syntax

```
SQLRETURN SQLNumResultCols (SQLHSTMT hstmt,  
                             SQLSMALLINT *pccol);
```

## Function arguments

Table 122. *SQLNumResultCols* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLSMALLINT *	<i>pccol</i>	Output	Number of columns in the result set.

## Usage

The function sets the output argument to zero if the last statement processed on the input statement handle is not a SELECT.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 123. *SQLNumResultCols* SQLSTATEs

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	<i>pcbCol</i> is a null pointer.
HY010	Function sequence error	The function is called before calling SQLPrepare or SQLExecDirect for the <i>hstmt</i> .
S1013 *	Memory management problem.	The driver is unable to access memory required to support the processing or completion of the function.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLColAttributes - Obtain column attributes” on page 52
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLGetCol - Retrieve one column of a row of the result set” on page 103
- “SQLPrepare - Prepare a statement” on page 164

### Related concepts

“Process UPDATE, DELETE, and INSERT statements in a DB2 UDB CLI application” on page 13

## SQLParamData - Get next parameter for which a data value is needed

### Purpose

SQLParamData() is used with SQLPutData() to send long data in pieces. It can also be used to send fixed length data.

### Syntax

```
SQLRETURN SQLParamData (SQLHSTMT hstmt,  
                        SQLPOINTER *prgbValue);
```

### Function arguments

Table 124. SQLParamData arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLPOINTER *	<i>prgbValue</i>	Output	Pointer to the value of the <i>prgbValue</i> argument specified on the SQLSetParam call.

### 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 defined value in *prgbValue* supplied by the application during the previous SQLBindParam() call. SQLPutData() is called one or more times 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 processed successfully. If any errors occur during or before actual statement processing, 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 fail. In addition, all function calls referencing the parent *hdbc* of *hstmt* fail if they involve changing any attribute or state of that connection. Those following function calls on the parent *hdbc* are also not permitted:

- SQLAllocConnect()
- SQLAllocHandle()
- SQLAllocStmt()
- SQLSetConnectOption()

Should they be called during an SQL\_NEED\_DATA sequence, these functions return SQL\_ERROR with SQLSTATE of HY010 and the processing of the SQL\_DATA\_AT\_EXEC parameters is not affected.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE
- SQL\_NEED\_DATA



## Diagnostics

SQLParamData() can return any SQLSTATE returned by the SQLExecDirect() and SQLExecute() functions. In addition, the following diagnostics can also be generated:

Table 125. SQLParamData SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>prgbValue</i> is a null pointer.
HY010	Function sequence error	SQLParamData() is called out of sequence. This call is only valid after an SQLExecDirect() or an SQLExecute(), or after an SQLPutData() call.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYDE0	No data at processing values pending	Even though this function is called after an SQLExecDirect() or an SQLExecute() call, there are no SQL_DATA_AT_EXEC parameters (remaining) to process.

## SQLParamOptions - Specify an input array for a parameter

### Purpose

SQLParamOptions() provides the ability to set multiple values for each parameter set by SQLBindParameter(). This allows the application to insert multiple rows into a table on a single call to SQLExecute() or SQLExecDirect().

### Syntax

```
SQLRETURN SQLParamOptions (SQLHSTMT StatementHandle,  
SQLINTEGER Crow,  
SQLINTEGER *FetchOffsetPtr);
```

### Function arguments

Table 126. SQLParamOptions arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLINTEGER	<i>Crow</i>	Input	Number of values for each parameter. If this is greater than 1, then the <i>rgbValue</i> argument in SQLBindParameter() points to an array of parameter values, and <i>pcbValue</i> points to an array of lengths.
SQLINTEGER *	<i>FetchOffsetPtr</i>	Output (deferred)	Not currently used.

### Usage

This function can be used with SQLBindParameter() to set up a multiple-row INSERT statement. In order to accomplish this, the application must allocate storage for all of the data being inserted. This data must be organized in a row-wise fashion. This means that all of the data for the first row is contiguous, followed by all the data for the next row, and so on. The SQLBindParameter() function should be used to

bind all of the input parameter types and lengths. In the case of a multiple-row INSERT statement, the addresses provided on `SQLBindParameter()` are used to reference the first row of data. All subsequent rows of data are referenced by incrementing those addresses by the length of the entire row.

For instance, the application intends to insert 100 rows of data into a table, and each row contains a 4-byte integer value, followed by a 10-byte character value. To do this, the application allocates 1400 bytes of storage, and fills each 14-byte piece of storage with the appropriate data for the row.

Also, the indicator pointer passed on the `SQLBindParameter()` must reference an 800-byte piece of storage. This is used to pass in any null indicator values. This storage is also row-wise, so the first 8 bytes are the 2 indicators for the first row, followed by the 2 indicators for the next row, and so on. The `SQLParamOptions()` function is used by the application to specify how many rows are inserted on the next processing of an INSERT statement using the statement handle. The INSERT statement must be of the multiple-row form. For example:

```
INSERT INTO CORPDATA.NAMES ? ROWS VALUES(?, ?)
```

## Return codes

- `SQL_SUCCESS`
- `SQL_SUCCESS_WITH_INFO`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

## Error conditions

Table 127. *SQLParamOptions SQLSTATEs*

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	The value in the argument <i>Crow</i> is less than 1.
HY010	Function sequence error	The function is called while in a data-at-processing ( <code>SQLParamData()</code> , <code>SQLPutData()</code> ) operation.

## Restrictions

None.

## References

- “`SQLBindParam` - Bind a buffer to a parameter marker” on page 38
- “`SQLMoreResults` - Determine whether there are more result sets” on page 155

## SQLPrepare - Prepare a statement

### Purpose

`SQLPrepare()` associates an SQL statement with the input statement handle and sends the statement to the DBMS to be prepared. The application can reference this prepared statement by passing the statement handle to other functions.

If the statement handle has been used with a SELECT statement, `SQLFreeStmt()` must be called to close the cursor, before calling `SQLPrepare()`.

### Syntax

```
SQLRETURN SQLPrepare (SQLHSTMT      hstmt,
                     SQLCHAR        *szSqlStr,
                     SQLINTEGER      cbSqlStr);
```

## Function arguments

Table 128. SQLPrepare arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szSqlStr</i>	Input	SQL statement string.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle. There must not be an open cursor associated with <i>hstmt</i> .
SQLINTEGER	<i>cbSqlStr</i>	Input	Length of contents of <i>szSqlStr</i> argument.  This must be set to either the exact length of the SQL statement in <i>szSqlStr</i> , or to SQL_NTS if the statement text is null-terminated.

## Usage

As soon as a statement has been prepared using SQLPrepare(), the application can request information about the format of the result set (if it is a SELECT statement) by calling:

- SQLNumResultCols()
- SQLDescribeCol()
- SQLColAttributes()

A prepared statement can be processed once, or multiple times by calling SQLExecute(). The SQL statement remains associated with the statement handle until the handle is used with another SQLPrepare(), SQLExecDirect(), SQLColumns(), SQLSpecialColumns(), SQLStatistics(), or SQLTables().

The SQL statement string might contain parameter markers. A parameter marker is represented by a "?" character, and indicates a position in the statement where the value of an application variable is to be substituted, when SQLExecute() is called. SQLBindParam() is used to bind (or associate) an application variable to each parameter marker, and to indicate if any data conversion should be performed at the time the data is transferred.

The SQL statement cannot be a COMMIT or ROLLBACK. SQLTransact() must be called to issue COMMIT or ROLLBACK.

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.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 129. SQLPrepare SQLSTATEs

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	There is an open cursor on the specified <i>hstmt</i> .

Table 129. SQLPrepare SQLSTATEs (continued)

SQLSTATE	Description	Explanation
37xxx	Syntax error or access violation	<i>szSqlStr</i> contained one or more of the following statements: <ul style="list-style-type: none"> <li>• A COMMIT</li> <li>• A ROLLBACK</li> <li>• An SQL statement that the connected database server cannot prepare</li> <li>• A statement containing a syntax error</li> </ul>
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	<i>szSqlStr</i> is a null pointer.  The argument <i>cbSqlStr</i> is less than 1, but not equal to SQL_NTS.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

**Note:** Not all Database Management Systems (DBMSs) report all of the above diagnostic messages at prepare time. Therefore an application must also be able to handle these conditions when calling SQLExecute().

## Example

Refer to “Example: Interactive SQL and the equivalent DB2 UDB CLI function calls” on page 253 for a listing of the `check_error`, `initialize`, and `terminate` functions used in the following example.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/*****
** file = prepare.c
**
** Example of preparing then repeatedly executing an SQL statement.
**
** Functions used:
**
**      SQLAllocConnect      SQLFreeConnect
**      SQLAllocEnv         SQLFreeEnv
**      SQLAllocStmt        SQLFreeStmt
**      SQLConnect          SQLDisconnect
**
**      SQLBindCol          SQLFetch
**      SQLTransact         SQLError
**      SQLPrepare          SQLSetParam
**      SQLExecute
*****/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255

int initialize(SQLHENV *henv,

```

```

        SQLHDBC *hdbc);

int terminate(SQLHENV henv,
             SQLHDBC hdbc);

int print_error (SQLHENV   henv,
                SQLHDBC   hdbc,
                SQLHSTMT  hstmt);

int check_error (SQLHENV   henv,
                SQLHDBC   hdbc,
                SQLHSTMT  hstmt,
                SQLRETURN  rc);

/*****
** main
** - initialize
** - terminate
*****/
int main()
{
    SQLHENV   henv;
    SQLHDBC   hdbc;
    SQLCHAR   sqlstmt[MAX_STMT_LEN + 1]="";
    SQLRETURN rc;

    rc = initialize(&henv, &hdbc);
    if (rc == SQL_ERROR) return(terminate(henv, hdbc));

    {SQLHSTMT  hstmt;
    SQLCHAR   sqlstmt []="SELECT deptname, location from org where division = ?";
    SQLCHAR   deptname[15],
              location[14],
              division[11];

    SQLINTEGER rlength,
              plength;

    rc = SQLAllocStmt(hdbc, &hstmt);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

    /* prepare statement for multiple use */
    rc = SQLPrepare(hstmt, sqlstmt, SQL_NTS);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    /* bind division to parameter marker in sqlstmt */
    rc = SQLSetParam(hstmt, 1, SQL_CHAR, SQL_CHAR, 10, 10, division,
                    &plength);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    /* bind deptname to first column in the result set */
    rc = SQLBindCol(hstmt, 1, SQL_CHAR, (SQLPOINTER) deptname, 15,
                    &rlength);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);
    rc = SQLBindCol(hstmt, 2, SQL_CHAR, (SQLPOINTER) location, 14,
                    &rlength);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    printf("\nEnter Division Name or 'q' to quit:\n");
    printf("(Eastern, Western, Midwest, Corporate)\n");
    gets(division);
    plength = SQL_NTS;

```

```

while(division[0] != 'q')
{
    rc = SQLExecute(hstmt);
    if (rc != SQL_SUCCESS )
        check_error (henv, hdbc, hstmt, rc);

    printf("Departments in %s Division:\n", division);
    printf("DEPTNAME      Location\n");
    printf("-----\n");

    while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS)
    {
        printf("%-14.14s %-13.13s \n", deptname, location);
    }
    if (rc != SQL_NO_DATA_FOUND )
        check_error (henv, hdbc, hstmt, rc);
    SQLFreeStmt(hstmt, SQL_CLOSE);
    printf("\nEnter Division Name or 'q' to quit:\n");
    printf("(Eastern, Western, Midwest, Corporate)\n");
    gets(division);
}

rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
if (rc != SQL_SUCCESS )
    check_error (henv, hdbc, SQL_NULL_HSTMT, rc);

terminate(henv, hdbc);
return (0);
}/* end main */

```

## References

- “SQLColAttributes - Obtain column attributes” on page 52
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLNumResultCols - Get number of result columns” on page 160

## SQLPrimaryKeys - Get primary key columns of a table

### Purpose

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 that is generated by a query.

### Syntax

```

SQLRETURN  SQLPrimaryKeys (SQLHSTMT          StatementHandle,
                          SQLCHAR           *CatalogName,
                          SQLSMALLINT      NameLength1,
                          SQLCHAR           *SchemaName,
                          SQLSMALLINT      NameLength2,
                          SQLCHAR           *TableName,
                          SQLSMALLINT      NameLength3);

```

## Function arguments

Table 130. *SQLPrimaryKeys* arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	Catalog qualifier of a 3 part table name.  This must be a NULL pointer or a zero length string.
SQLCHAR *	<i>SchemaName</i>	Input	Schema qualifier of table name.
SQLCHAR *	<i>TableName</i>	Input	Table name.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT	<i>NameLength2</i>	Input	Length of <i>SchemaName</i> .
SQLSMALLINT	<i>NameLength3</i>	Input	Length of <i>TableName</i> .
SQLSMALLINT	<i>NameLength1</i>	Input	Length of <i>CatalogName</i> .

## Usage

`SQLPrimaryKeys()` returns the primary key columns from a single table. Search patterns cannot be used to specify the schema qualifier or the table name.

The result set contains the columns that are listed in Table 131, ordered by `TABLE_CAT`, `TABLE_SCHEM`, `TABLE_NAME`, and `ORDINAL_POSITION`.

Because 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.

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

Table 131. *Columns returned by SQLPrimaryKeys*

Column number/name	Data type	Description
1 TABLE_CAT	VARCHAR (128)	The current server.
2 TABLE_SCHEM	VARCHAR (128)	The name of the schema containing TABLE_NAME.
3 TABLE_NAME	VARCHAR (128) not NULL	Name of the specified table.
4 COLUMN_NAME	VARCHAR (128) not NULL	Primary Key column name.
5 ORDINAL_POSITION	SMALLINT not NULL	Column sequence number in the primary key, starting with 1.
6 PK_NAME	VARCHAR(128)	Primary key identifier. NULL if not applicable to the data source.

**Note:** The column names used by DB2 UDB 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

## Error conditions

Table 132. SQLPrimaryKeys SQLSTATES

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	A cursor is already opened on the statement handle.
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY008	Operation canceled	
HY010	Function sequence error	The function is called while in a data-at-processing (SQLParamData(), SQLPutData()) operation.
HY014	No more handles	DB2 UDB CLI is unable to allocate a handle due to internal resources.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid .
HY090	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal to SQL_NTS.
HYC00	Driver not capable	DB2 UDB CLI does not support <i>catalog</i> as a qualifier for table name.
HYT00	Timeout expired	

## Restrictions

None.

## References

- “SQLForeignKeys - Get the list of foreign key columns” on page 94
- “SQLStatistics - Get index and statistics information for a base table” on page 209

## SQLProcedureColumns - Get input/output parameter information for a procedure

### Purpose

SQLProcedureColumns() returns a list of input and output parameters associated with a 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 that is generated by a query.

### Syntax

```
SQLRETURN SQLProcedureColumns(SQLHSTMT          StatementHandle,
                               SQLCHAR          *CatalogName,
                               SQLSMALLINT      NameLength1,
                               SQLCHAR          *SchemaName,
                               SQLSMALLINT      NameLength2,
                               SQLCHAR          *ProcName,
                               SQLSMALLINT      NameLength3,
                               SQLCHAR          *ColumnName,
                               SQLSMALLINT      NameLength4);
```



## Function arguments

Table 133. *SQLProcedureColumns* arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	Catalog qualifier of a 3 part procedure name.  This must be a NULL pointer or a zero length string.
SQLCHAR *	<i>ColumnName</i>	Input	Buffer that might contain a <i>pattern-value</i> 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 or SchemaName.
SQLCHAR *	<i>ProcName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by procedure name.
SQLCHAR *	<i>SchemaName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by schema name.  For DB2 Universal Database for z/OS and OS/390® V 4.1, all the stored procedures are in one schema; the only acceptable value for the <i>SchemaName</i> argument is a null pointer. For DB2 Universal Database, <i>SchemaName</i> can contain a valid pattern value.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT	<i>NameLength1</i>	Input	Length of <i>CatalogName</i> . This must be set to 0.
SQLSMALLINT	<i>NameLength2</i>	Input	Length of <i>SchemaName</i> .
SQLSMALLINT	<i>NameLength3</i>	Input	Length of <i>ProcName</i> .
SQLSMALLINT	<i>NameLength4</i>	Input	Length of <i>ColumnName</i> .

## Usage

DB2 UDB CLI returns information about the input, input and output, and output parameters associated with the stored procedure, but cannot return information about the descriptor for any result sets returned.

`SQLProcedureColumns()` returns the information in a result set, ordered by `PROCEDURE_CAT`, `PROCEDURE_SCHEM`, `PROCEDURE_NAME`, and `COLUMN_TYPE`. Table 134 lists the columns in the result set. Applications should be aware that columns beyond the last column might be defined in future releases.

Because 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.

Table 134. *Columns returned by SQLProcedureColumns*

Column number/name	Data type	Description
1 PROCEDURE_CAT	VARCHAR(128)	The current server.
2 PROCEDURE_SCHEM	VARCHAR(128)	The name of the schema containing PROCEDURE_NAME.
3 PROCEDURE_NAME	VARCHAR(128)	Name of the procedure.

Table 134. Columns returned by SQLProcedureColumns (continued)

Column number/name	Data type	Description
4 COLUMN_NAME	VARCHAR(128)	Name of the parameter.
5 COLUMN_TYPE	SMALLINT not NULL	Identifies the type information associated with this row. The values can be: <ul style="list-style-type: none"> <li>SQL_PARAM_TYPE_UNKNOWN – the parameter type is unknown. <b>Note:</b> This is not returned.</li> <li>SQL_PARAM_INPUT – this parameter is an input parameter.</li> <li>SQL_PARAM_INPUT_OUTPUT – this parameter is an input / output parameter.</li> <li>SQL_PARAM_OUTPUT – this parameter is an output parameter.</li> <li>SQL_RETURN_VALUE – the procedure column is the return value of the procedure. <b>Note:</b> This is not returned.</li> <li>SQL_RESULT_COL – this parameter is actually a column in the result set. <b>Note:</b> This is not returned.</li> </ul>
6 DATA_TYPE	SMALLINT not NULL	SQL data type.
7 TYPE_NAME	VARCHAR(128) not NULL	Character string representing the name of the data type corresponding to DATA_TYPE.
8 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 bytes 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.
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.
10 DECIMAL_DIGITS	SMALLINT	The scale of the parameter. NULL is returned for data types where scale is not applicable.

Table 134. Columns returned by SQLProcedureColumns (continued)

Column number/name	Data type	Description
11 NUM_PREC_RADIX	SMALLINT	<p>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.</p> <p>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.</p> <p>For numeric data types, the Database Management System (DBMS) can return a NUM_PREC_RADIX of either 10 or 2.</p> <p>NULL is returned for data types where radix is not applicable.</p>
12 NULLABLE	VARCHAR(3)	<p>'NO' if the parameter does not accept NULL values.</p> <p>'YES' if the parameter accepts NULL values.</p>
13 REMARKS	VARCHAR(254)	Might contain descriptive information about the parameter.
14 COLUMN_DEF	VARCHAR	<p>The default value of the column.</p> <p>If NULL is 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 is specified, then this column is NULL.</p> <p>The value of COLUMN_DEF can be used in generating a new column definition, except when it contains the value TRUNCATED.</p>
15 SQL_DATA_TYPE	SMALLINT not NULL	<p>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 UDB CLI does not support interval data types).</p> <p>For datetime data types, the SQL_DATA_TYPE field in the result set is SQL_DATETIME, and the SQL_DATETIME_SUB field returns the subcode for the specific datetime data type (SQL_CODE_DATE, SQL_CODE_TIME or SQL_CODE_TIMESTAMP).</p>
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 UDB CLI does not support).
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.

Table 134. Columns returned by *SQLProcedureColumns* (continued)

Column number/name	Data type	Description
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.
19 IS_NULLABLE	VARCHAR	<ul style="list-style-type: none"> <li>• “NO” if the column does not include NULLs.</li> <li>• “YES” if the column can include NULLs.</li> <li>• zero-length string if nullability is unknown.</li> </ul> <p>ISO rules are followed to determine nullability.</p> <p>An ISO SQL-compliant DBMS cannot return an empty string.</p> <p>The value returned for this column is different than the value returned for the NULLABLE column. (See the description of the NULLABLE column.)</p>

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_STILL\_EXECUTING
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Error conditions

Table 135. *SQLProcedureColumns SQLSTATEs*

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	A cursor is already opened on the statement handle.
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
42601	PARMLIST syntax error	The PARMLIST value in the stored procedures catalog table contains a syntax error.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY008	Operation canceled	
HY010	Function sequence error	
HY014	No more handles	DB2 UDB CLI is unable to allocate a handle due to internal resources.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal SQL_NTS.
HYC00	Driver not capable	<p>DB2 UDB CLI does not support <i>catalog</i> as a qualifier for procedure name.</p> <p>The connected server does not support <i>schema</i> as a qualifier for procedure name.</p>

Table 135. *SQLProcedureColumns SQLSTATES (continued)*

SQLSTATE	Description	Explanation
HYT00	Timeout expired	

## Restrictions

SQLProcedureColumns() does not return information about the attributes of result sets that can 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() returns an empty result set.

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/* From CLI sample proccols.c */
/* ... */

printf("Enter Procedure Schema Name Search Pattern:\n");
gets((char *)proc_schem.s);

printf("Enter Procedure Name Search Pattern:\n");
gets((char *)proc_name.s);

rc = SQLProcedureColumns(hstmt, NULL, 0, proc_schem.s, SQL_NTS,
                        proc_name.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) proc_schem.s, 129,
                &proc_schem.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) proc_name.s, 129,
                &proc_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 4, SQL_C_CHAR, (SQLPOINTER) column_name.s, 129,
                &column_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 5, SQL_C_SHORT, (SQLPOINTER) &arg_type,
                0, &arg_type_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) type_name.s, 129,
                &type_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 8, SQL_C_LONG, (SQLPOINTER) &length,
                0, &length_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 10, SQL_C_SHORT, (SQLPOINTER) &scale,
                0, &scale_ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 13, SQL_C_CHAR, (SQLPOINTER) remarks.s, 255,
                &remarks.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

```

```

/* Fetch each row, and display */
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    sprintf((char *)cur_name, "%s.%s", proc_schem.s, proc_name.s);
    if (strcmp((char *)cur_name, (char *)pre_name) != 0) {
        printf("\n%s\n", cur_name);
    }
    strcpy((char *)pre_name, (char *)cur_name);
    printf("  %s", column_name.s);
    switch (arg_type)
    { case SQL_PARAM_INPUT : printf(", Input"); break;
      case SQL_PARAM_OUTPUT : printf(", Output"); break;
      case SQL_PARAM_INPUT_OUTPUT : printf(", Input_Output"); break;
    }
    printf(", %s", type_name.s);
    printf(" (%ld", length);
    if (scale_ind != SQL_NULL_DATA) {
        printf(", %d)\n", scale);
    } else {
        printf(")\n");
    }
    if (remarks.ind > 0) {
        printf("(remarks), %s)\n", remarks.s);
    }
}
/* endwhile */

```

## References

- “SQLProcedures - Get list of procedure names”

## SQLProcedures - Get list of procedure names

### Purpose

SQLProcedures() returns a list of 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 that is generated by a query.

### Syntax

```

SQLRETURN  SQLProcedures  (SQLHSTMT      StatementHandle,
                           SQLCHAR       *CatalogName,
                           SQLSMALLINT   NameLength1,
                           SQLCHAR       *SchemaName,
                           SQLSMALLINT   NameLength2,
                           SQLCHAR       *ProcName,
                           SQLSMALLINT   NameLength3);

```

### Function arguments

Table 136. SQLProcedures arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>CatalogName</i>	Input	Catalog qualifier of a 3 part procedure name. This must be a NULL pointer or a zero length string.
SQLCHAR *	<i>ProcName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by procedure name.

Table 136. SQLProcedures arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>SchemaName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by schema name.  For DB2 Universal Database for z/OS and OS/390 V 4.1, all the stored procedures are in one schema; the only acceptable value for the <i>SchemaName</i> argument is a null pointer. For DB2 Universal Database, <i>SchemaName</i> can contain a valid pattern value.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT	<i>NameLength2</i>	Input	Length of <i>SchemaName</i> .
SQLSMALLINT	<i>NameLength3</i>	Input	Length of <i>ProcName</i> .
SQLSMALLINT	<i>NameLength1</i>	Input	Length of <i>CatalogName</i> . This must be set to 0.

## Usage

The result set returned by SQLProcedures() contains the columns listed in Table 137 in the order given. The rows are ordered by PROCEDURE\_CAT, PROCEDURE\_SCHEMA, and PROCEDURE\_NAME.

Because calls to SQLProcedures() in many cases map to a complex and thus expensive query against the system catalog, use them sparingly, and save the results rather than repeating calls.

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

Table 137. Columns returned by SQLProcedures

Column number/name	Data type	Description
PROCEDURE_CAT	VARCHAR(128)	The current server.
PROCEDURE_SCHEM	VARCHAR(128)	The name of the schema containing PROCEDURE_NAME.
PROCEDURE_NAME	VARCHAR(128) NOT NULL	The name of the procedure.
NUM_INPUT_PARAMS	INTEGER not NULL	Number of input parameters.  This column should not be used, it is reserved for future use by ODBC.  It is used in versions of DB2 UDB CLI before version 5. For backward compatibility it can be used with the old DB2CLI.PROCEDURES pseudo catalog table (by setting the PATCH1 CLI/ODBC Configuration keyword).
NUM_OUTPUT_PARAMS	INTEGER not NULL	Number of output parameters.  This column should not be used, it is reserved for future use by ODBC.  It was used in versions of DB2 UDB CLI before version 5. For backward compatibility it can be used with the old DB2CLI.PROCEDURES pseudo catalog table (by setting the PATCH1 CLI/ODBC Configuration keyword).

Table 137. Columns returned by SQLProcedures (continued)

Column number/name	Data type	Description
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.  It was used in versions of DB2 UDB CLI before version 5. For backward compatibility it can be used with the old DB2CLI.PROCEDURES pseudo catalog table (by setting the PATCH1 CLI/ODBC Configuration keyword).
REMARKS	VARCHAR(254)	Contains the descriptive information about the procedure.

**Note:** The column names used by DB2 UDB 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

### Error conditions

Table 138. SQLProcedures SQLSTATEs

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	A cursor is already opened on the statement handle.
40003 08S01	Communication link failure	The communication link between the application and data source fails before the function is completed.
HY001	Memory allocation failure	DB2 UDB CLI is unable to allocate memory required to support the processing or completion of the function.
HY008	Operation canceled	
HY010	Function sequence error	
HY014	No more handles	DB2 UDB CLI is unable to allocate a handle due to internal resources.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HY090	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal to SQL_NTS.
HYC00	Driver not capable	DB2 UDB CLI does not support <i>catalog</i> as a qualifier for procedure name.  The connected server does not support schema as a qualifier for procedure name.
HYT00	Timeout expired	



## 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()` returns an empty result set.

## Example

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```
/* From CLI sample procs.c */
/* ... */

printf("Enter Procedure Schema Name Search Pattern:\n");
gets((char *)proc_schem.s);

rc = SQLProcedures(hstmt, NULL, 0, proc_schem.s, SQL_NTS, (SQLCHAR *)"%", SQL_NTS);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 2, SQL_C_CHAR, (SQLPOINTER) proc_schem.s, 129,
                &proc_schem.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 3, SQL_C_CHAR, (SQLPOINTER) proc_name.s, 129,
                &proc_name.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

rc = SQLBindCol(hstmt, 7, SQL_C_CHAR, (SQLPOINTER) remarks.s, 255,
                &remarks.ind);
CHECK_HANDLE( SQL_HANDLE_STMT, hstmt, rc );

printf("PROCEDURE SCHEMA          PROCEDURE NAME          \n");
printf("----- \n");
/* Fetch each row, and display */
while ((rc = SQLFetch(hstmt)) == SQL_SUCCESS) {
    printf("%-25s %-25s\n", proc_schem.s, proc_name.s);
    if (remarks.ind != SQL_NULL_DATA) {
        printf(" (Remarks) %s\n", remarks.s);
    }
}
/* endwhile */
```

## References

- “SQLProcedureColumns - Get input/output parameter information for a procedure” on page 170

## SQLPutData - Pass data value for a parameter

### Purpose

`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      hstmt,
                     SQLPOINTER    rgbValue,
                     SQLINTEGER     cbValue);
```

## Function arguments

Table 139. SQLPutData arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER	<i>cbValue</i>	Input	<p>Length of <i>rgbValue</i>. Specifies the amount of data sent in a call to SQLPutData().</p> <p>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 <i>cbValue</i>.</p> <p><i>cbValue</i> is ignored for all date, time, timestamp data types, and all numeric data types except SQL_NUMERIC and SQL_DECIMAL.</p> <p>For cases where the C buffer type is SQL_CHAR or SQL_BINARY, or if SQL_DEFAULT is specified as the C buffer type and the C buffer type default is SQL_CHAR or SQL_BINARY, this is the number of bytes of data in the <i>rgbValue</i> buffer.</p>
SQLPOINTER	<i>rgbValue</i>	Input	Pointer to the actual data, or portion of data, for a parameter. The data must be in the form specified in the SQLBindParam() call that the application used when specifying the parameter.

## 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 through repeated calls to SQLPutData(). After all the pieces of data for the parameter have been sent, the application again calls SQLParamData(). SQLParamData() proceeds to the next SQL\_DATA\_AT\_EXEC parameter, or, if all parameters have data values, executes the statement.

SQLPutData() cannot be called more than once for a fixed length parameter.

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 fail. In addition, all function calls referencing the parent *hdbc* of *hstmt* fail if they involve changing any attribute or state of that connection. For a list of these functions, see the Usage section for "SQLParamData - Get next parameter for which a data value is needed" on page 162.

If one or more calls to SQLPutData() for a single parameter result in SQL\_SUCCESS, attempting to call SQLPutData() with *cbValue* set to SQL\_NULL\_DATA for the same parameter results in an error with SQLSTATE of HY011. 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\_ERROR

- SQL\_INVALID\_HANDLE

## Diagnostics

Some of the following diagnostics conditions might be reported on the final SQLParamData() call rather than at the time the SQLPutData() is called.

Table 140. SQLPutData SQLSTATEs

SQLSTATE	Description	Explanation
22001	Too much data	The size of the data supplied to the current parameter by SQLPutData() exceeds the size of the parameter. The data supplied by the last call to SQLPutData() is ignored.
01004	Data truncated	The data sent for a numeric parameter is truncated without the loss of significant digits.  Timestamp data sent for a date or time column is truncated.  Function returns with SQL_SUCCESS_WITH_INFO.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	The argument <i>rgbValue</i> is a null pointer.  The argument <i>rgbValue</i> is not a NULL pointer and the argument <i>cbValue</i> is less than 0, but not equal to SQL_NTS or SQL_NULL_DATA.
HY010	Function sequence error	The statement handle <i>hstmt</i> must be in a <i>need data</i> state and must have been positioned on an SQL_DATA_AT_EXEC parameter through a previous SQLParamData() call.

## SQLReleaseEnv - Release all environment resources

### Purpose

SQLReleaseEnv() invalidates and frees the environment handle. All DB2 UDB CLI resources associated with the environment handle are freed.

SQLFreeConnect() must be called before calling this function.

This function is the last DB2 UDB CLI step an application needs to do before terminating.

### Syntax

```
SQLRETURN SQLReleaseEnv (SQLHENV henv);
```

### Function arguments

Table 141. SQLReleaseEnv arguments

Data type	Argument	Use	Description
SQLHENV	<i>henv</i>	Input	Environment handle.

## Usage

If this function is called when there is still a valid connection handle, `SQL_ERROR` is returned, and the environment handle remains valid.

## Return codes

- `SQL_SUCCESS`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

## Diagnostics

Table 142. *SQLReleaseEnv* SQLSTATEs

SQLSTATE	Description	Explanation
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY010	Function sequence error	There is an <i>hdbc</i> which is in allocated or connected state. Call <code>SQLDisconnect</code> and <code>SQLFreeConnect</code> for the <i>hdbc</i> before calling <code>SQLReleaseEnv</code> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## Example

Refer to the example in the “`SQLAllocEnv` - Allocate environment handle” on page 24.

## References

- “`SQLFreeConnect` - Free connection handle” on page 99

## SQLRowCount - Get row count

### Purpose

`SQLRowCount()` returns the number of rows in a table affected by an `UPDATE`, `INSERT`, or `DELETE` statement processed against the table, or a view based on the table.

`SQLExecute()` or `SQLExecDirect()` must be called before calling this function.

### Syntax

```
SQLRETURN SQLRowCount (SQLHSTMT      hstmt,  
                      SQLINTEGER     *pcrow);
```

### Function arguments

Table 143. *SQLRowCount* arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER *	<i>pcrow</i>	Output	Pointer to location where the number of rows affected is stored.

## Usage

If the last processed statement referenced by the input statement handle is not an UPDATE, INSERT, or DELETE statement, or if it is not processed successfully, then the function sets the contents of *pcrow* to 0.

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\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 144. SQLRowCount SQLSTATES

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	<i>pcrow</i> is a null pointer.
HY010	Function sequence error	The function is called before calling SQLExecute or SQLExecDirect for the <i>hstmt</i> .
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## References

- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLNumResultCols - Get number of result columns” on page 160

## SQLSetConnectAttr - Set a connection attribute

### Purpose

SQLSetConnectAttr() sets connection attributes for a particular connection.

### Syntax

```
SQLRETURN SQLSetConnectAttr (SQLHDBC    hdbc,  
                             SQLINTEGER  fAttr,  
                             SQLPOINTER vParam,  
                             SQLINTEGER  sLen);
```

## Function arguments

Table 145. SQLSetConnectAttr arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Connection handle.
SQLINTEGER	<i>fAttr</i>	Input	Connect attribute to set, refer to Table 146 for more information.
SQLPOINTER	<i>vParam</i>	Input	Value associated with <i>fAttr</i> . Depending on the option, this can be a pointer to a 32-bit integer value, or a character string.
SQLINTEGER	<i>sLen</i>	Input	Length of input value, if it is a character string; otherwise, unused.

## Usage

All connection and statement options set through the SQLSetConnectAttr() persist until SQLFreeConnect() is called or the next SQLSetConnectAttr() call.

The format of information set through *vParam* depends on the specified *fAttr*. The option information can be either a 32-bit integer or a pointer to a null-terminated character string.

Table 146. Connect options

<i>fAttr</i>	Contents
SQL_2ND_LEVEL_TEXT	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_TRUE – Error text obtained by calling SQLError() contains the complete text description of the error.</li> <li>• SQL_FALSE – Error text obtained by calling SQLError() contains the first-level description of the error only. This is the default.</li> </ul>
SQL_ATTR_AUTOCOMMIT	<p>A 32-bit value that sets the commit behavior for the connection. The following are possible values:</p> <ul style="list-style-type: none"> <li>• SQL_TRUE – Each SQL statement is automatically committed as it is processed.</li> <li>• SQL_FALSE – The SQL statements are not automatically committed. If running with commitment control, changes must be explicitly committed or rolled back using either SQLEndTran() or SQLTransact(). This is the default.</li> </ul>

Table 146. Connect options (continued)

<i>fAttr</i>	Contents
<p>SQL_ATTR_COMMIT or SQL_TXN_ISOLATION</p>	<p>A 32-bit value that sets the transaction-isolation level for the current connection referenced by <i>hdbc</i>. The following values are accepted by DB2 UDB CLI, but each server might only support some of these isolation levels:</p> <ul style="list-style-type: none"> <li>• SQL_TXN_NO_COMMIT – Commitment control is not used.</li> <li>• SQL_TXN_READ_UNCOMMITTED – Dirty reads, nonrepeatable reads, and phantoms are possible.</li> <li>• SQL_TXN_READ_COMMITTED – Dirty reads are not possible. Non-repeatable reads and phantoms are possible.</li> <li>• SQL_TXN_REPEATABLE_READ – Dirty reads and nonrepeatable reads are not possible. Phantoms are possible.</li> <li>• SQL_TXN_SERIALIZABLE – Transactions are serializable. Dirty reads, non-repeatable reads, and phantoms are not possible.</li> </ul> <p>In IBM terminology,</p> <ul style="list-style-type: none"> <li>• SQL_TXN_READ_UNCOMMITTED is uncommitted read</li> <li>• SQL_TXN_READ_COMMITTED is cursor stability</li> <li>• SQL_TXN_REPEATABLE_READ is read stability</li> <li>• SQL_TXN_SERIALIZABLE is repeatable read</li> </ul> <p>For a detailed explanation of isolation levels, refer to the IBM DB2 SQL Reference.</p> <p>The SQL_ATTR_COMMIT attribute should be set before the SQLConnect(). If the value is changed after the connection has been established, and the connection is to a remote data source, the change does not take effect until the next successful SQLConnect() for the connection handle.</p>
<p>SQL_ATTR_DATE_FMT</p>	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_FMT_ISO – The International Organization for Standardization (ISO) date format yyyy-mm-dd is used. This is the default.</li> <li>• SQL_FMT_USA – The United States date format mm/dd/yyyy is used.</li> <li>• SQL_FMT_EUR – The European date format dd.mm.yyyy is used.</li> <li>• SQL_FMT_JIS – The Japanese Industrial Standard date format yyyy-mm-dd is used.</li> <li>• SQL_FMT_MDY – The date format mm/dd/yy is used.</li> <li>• SQL_FMT_DMY – The date format dd/mm/yy is used.</li> <li>• SQL_FMT_YMD – The date format yy/mm/dd is used.</li> <li>• SQL_FMT_JUL – The Julian date format yy/ddd is used.</li> <li>• SQL_FMT_JOB – The job default is used.</li> </ul>

Table 146. Connect options (continued)

fAttr	Contents
SQL_ATTR_DATE_SEP	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_SEP_SLASH – A slash ( / ) is used as the date separator. This is the default.</li> <li>• SQL_SEP_DASH – A dash ( - ) is used as the date separator.</li> <li>• SQL_SEP_PERIOD – A period ( . ) is used as the date separator.</li> <li>• SQL_SEP_COMMA – A comma ( , ) is used as the date separator.</li> <li>• SQL_SEP_BLANK – A blank is used as the date separator.</li> <li>• SQL_SEP_JOB – The job default is used.</li> </ul>
SQL_ATTR_DBC_DEFAULT_LIB	<p>A character value that indicates the default library that is used for resolving unqualified file references. This is not valid if the connection is using system naming mode.</p>
SQL_ATTR_DBC_SYS_NAMING	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_TRUE – DB2 UDB CLI uses the iSeries system naming mode. Files are qualified using the slash (/) delimiter. Unqualified files are resolved using the library list for the job.</li> <li>• SQL_FALSE – DB2 UDB CLI uses the default naming mode, which is SQL naming. Files are qualified using the period (.) delimiter. Unqualified files are resolved using either the default library or the current user ID.</li> </ul>
SQL_ATTR_DECIMAL_SEP	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_SEP_PERIOD – A period ( . ) is used as the decimal separator. This is the default.</li> <li>• SQL_SEP_COMMA – A comma ( , ) is used as the decimal separator.</li> <li>• SQL_SEP_JOB – The job default is used.</li> </ul>
SQL_ATTR_EXTENDED_COL_INFO	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_TRUE – Statement handles allocated against this connection handle can be used on SQLColAttributes() to retrieve extended column information, such as base table, base schema, base column, and label.</li> <li>• SQL_FALSE – Statement handles allocated against this connection handle cannot be used on the SQLColAttributes() function to retrieve extended column information. This is the default.</li> </ul>
SQL_ATTR_HEX_LITERALS	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_HEX_IS_CHAR – Hexadecimal constants are treated as character data. This is the default.</li> <li>• SQL_HEX_IS_BINARY – Hexadecimal constants are treated as binary data.</li> </ul>
SQL_ATTR_MAX_PRECISION	<p>An integer constant that is the maximum precision (length) that should be returned for the result data types. The value can be 31 or 63.</p>



Table 146. Connect options (continued)

<i>fAttr</i>	Contents
SQL_ATTR_MAX_SCALE	An integer constant that is the maximum scale (number of decimal positions to the right of the decimal point) that should be returned for the result data types. The value can range from 0 to the maximum precision.
SQL_ATTR_MIN_DIVIDE_SCALE	Specify the minimum divide scale (number of decimal positions to the right of the decimal point) that should be returned for the result data types resulting from a divide operation. The value can range from 0 to 9, not to exceed the maximum scale. If 0 is specified, minimum divide scale is not used.
SQL_ATTR_QUERY_OPTIMIZE_GOAL	<p>A 32-bit integer value that tells the optimizer to behave in a specified way when processing a query:</p> <ul style="list-style-type: none"> <li>• SQL_FIRST_IO – All queries are optimized with the goal of returning the first page of output as fast as possible. This goal works well when the output is controlled by a user who is most likely to cancel the query after viewing the first page of output data. Queries coded with an OPTIMIZE FOR nnn ROWS clause honor the goal specified by the clause.</li> <li>• SQL_ALL_IO – All queries are optimized with the goal of running the entire query to completion in the shortest amount of elapsed time. This is a good option when the output of a query is being written to a file or report, or the interface is queuing the output data. Queries coded with an OPTIMIZE FOR nnn ROWS clause honor the goal specified by the clause. This is the default.</li> </ul>
SQL_ATTR_TIME_FMT	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_FMT_ISO – The International Organization for Standardization (ISO) time format hh.mm.ss is used. This is the default.</li> <li>• SQL_FMT_USA – The United States time format hh:mmxx is used, where xx is AM or PM.</li> <li>• SQL_FMT_EUR – The European time format hh.mm.ss is used.</li> <li>• SQL_FMT_JIS – The Japanese Industrial Standard time format hh:mm:ss is used.</li> <li>• SQL_FMT_HMS – The hh:mm:ss format is used.</li> </ul>
SQL_ATTR_TIME_SEP	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_SEP_COLON – A colon ( : ) is used as the time separator. This is the default.</li> <li>• SQL_SEP_PERIOD – A period ( . ) is used as the time separator.</li> <li>• SQL_SEP_COMMA – A comma ( , ) is used as the time separator.</li> <li>• SQL_SEP_BLANK – A blank is used as the time separator.</li> <li>• SQL_SEP_JOB – The job default is used.</li> </ul>

| *Table 146. Connect options (continued)*

<i>fAttr</i>	<b>Contents</b>
SQL_ATTR_TXN_EXTERNAL	<p>A 32-bit integer value that must be SQL_TRUE to enable the use of XA transaction setting in the CLI connection. SQL_ATTR_TXN_EXTERNAL must be set to SQL_TRUE to use the XA transaction options by the SQL_ATTR_TXN_INFO connection attribute.</p> <p>The default is SQL_FALSE, which is not to enable XA transaction support. However, as soon as transaction support is enabled for the connection, it cannot be disabled. (Attempting to set SQL_ATTR_TXN_EXTERNAL to SQL_FALSE results in a CLI error.)</p> <p>Further information as well as an example of use of the SQL_ATTR_TXN_EXTERNAL connection attribute can be found in "Example: Use the CLI XA transaction connection attributes" on page 250.</p>

Table 146. Connect options (continued)

fAttr	Contents
SQL_ATTR_TXN_INFO	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_TXN_CREATE – Create and start a transaction. This parallels the xa_start(TMNOFLAGS) XA option.</li> <li>• SQL_TXN_END – End the specified transaction. The user is responsible to commit or roll back the work. This parallels the xa_end(TMSUCCESS) XA option.</li> <li>• SQL_TXN_END_FAIL – End the specified transaction and mark the transaction as rollback required. This parallels the xa_end(TMFAIL) XA option.</li> <li>• SQL_TXN_CLEAR – Suspend the transaction to work on a different transaction. This parallels the xa_end(TMSUSPEND) XA option.</li> <li>• SQL_TXN_FIND – Find, retrieve, and use the nonsuspended transaction specified in vParam for the current connection. This allows work to continue on the open cursors for the previously nonsuspended transaction. This parallels the xa_start(TMJOIN) XA option.</li> <li>• SQL_TXN_RESUME – Find, retrieve, and use the suspended transaction specified in vParam for the current connection. This allows work to continue on the open cursors for the previously suspended transaction. This parallels the xa_start(TMRESUME) XA option.</li> </ul> <p>Use of this connection attribute requires the user to be running in server mode. Keep in mind, a user cannot toggle between a non-server mode and server mode environment.</p> <p>The input argument vParam must point to a TXN_STRUCT object. This structure can be found in the header file QSYSINC/h.SQLCLI.</p> <p>The xa_info argument for the xa_open XA API must include the THDCTL=C keyword and value when using CLI with XA transactions.</p> <p>See XA transaction support for commitment control in the Commitment control topic for more information about XA transactions.</p> <p>See XA API for more information.</p> <p>See “Example: Use the CLI XA transaction connection attributes” on page 250 for more information and an example that shows how you can use the SQL_ATTR_TXN_INFO connection attribute.</p> <p>When running XA calls through CLI, the return codes from CLI reflect the XA return code specifications. These values can be found in the XA specification documentation, as well as in the XA.h include file. Note that the return code values that are listed in the XA include file take precedence over the CLI return code values when calling XA through this connection attribute.</p>

Table 146. Connect options (continued)

<i>fAttr</i>	Contents
SQL_ATTR_UCS2	A 32-bit integer value: <ul style="list-style-type: none"> <li>SQL_TRUE – When using statement handles allocated against this connection handle for SQLPrepare() and SQLExecDirect(), the statement text is passed in the UCS-2 (Unicode) coded character set identifier (CCSID).</li> <li>SQL_FALSE – When using statement handles allocated against this connection handle for SQLPrepare() and SQLExecDirect(), the statement text is passed in the job's CCSID. This is the default.</li> </ul>
SQL_SAVEPOINT_NAME	A character value that indicates the savepoint name to be used by SQLEndTran() on the functions SQL_SAVEPOINT_NAME_ROLLBACK or SQL_SAVEPOINT_NAME_RELEASE.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 147. SQLSetConnectAttr SQLSTATES

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	Given the <i>fAttr</i> value, a value that is not valid is specified for the argument <i>vParam</i> .  An <i>fAttr</i> that is not valid value is specified.

#### Related reference

“SQLSetStmtOption - Set statement option” on page 205

“SQLSetConnectOption - Set connection option”

## SQLSetConnectOption - Set connection option

### Purpose

SQLSetConnectOption() has been deprecated and replaced with SQLSetConnectAttr(). Although this version of DB2 UDB CLI continues to support SQLSetConnectOption(), it is recommended that you begin using SQLSetConnectAttr() in your DB2 UDB CLI programs so that they conform to the latest standards.

SQLSetConnectOption() sets connection attributes for a particular connection.

### Syntax

```
SQLRETURN SQLSetConnectOption (SQLHDBC hdbc,
                               SQLSMALLINT fOption,
                               SQLPOINTER vParam);
```

## Function arguments

Table 148. *SQLSetConnectOption* arguments

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Connection handle.
SQLPOINTER	<i>vParam</i>	Input	Value associated with <i>fOption</i> . Depending on the option, this can be a pointer to a 32-bit integer value, or a character string.
SQLSMALLINT	<i>fOption</i>	Input	Connect option to set, refer to Table 146 on page 184 for more information.

## Usage

The `SQLSetConnectOption()` provides many of the same attribute functions as `SQLSetConnectAttr()` prior to V5R3. However, `SQLSetConnectOption()` has since been deprecated, and support for all new attribute functions has gone into `SQLSetConnectAttr()`. Users should migrate to the nondeprecated interface.

All connection and statement options set through the `SQLSetConnectOption()` persist until `SQLFreeConnect()` is called or the next `SQLSetConnectOption()` call.

The format of information set through *vParam* depends on the specified *fOption*. The option information can be either a 32-bit integer or a pointer to a null-terminated character string.

Refer to Table 146 on page 184 for the appropriate connect options.

**Note:** Because `SQLSetConnectOption()` has been deprecated, not all the options listed in the table are supported.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 149. *SQLSetConnectOption* SQLSTATES

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	Given the <i>fOption</i> value, a value that is not valid is specified for the argument <i>vParam</i> .  A <i>fOption</i> value that is not valid is specified.
HYC00	Driver not capable	The specified <i>fOption</i> is not supported by DB2 UDB CLI or the server.  Given specified <i>fOption</i> value, the value specified for the argument <i>vParam</i> is not supported.

## Related reference

“SQLSetConnectAttr - Set a connection attribute” on page 183

## SQLSetCursorName - Set cursor name

### Purpose

SQLSetCursorName() associates a cursor name with the statement handle. This function is optional because DB2 UDB CLI implicitly generates a cursor name when needed.

### Syntax

```
SQLRETURN SQLSetCursorName (SQLHSTMT      hstmt,  
                             SQLCHAR      *szCursor,  
                             SQLSMALLINT  cbCursor);
```

### Function arguments

Table 150. SQLSetCursorName arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szCursor</i>	Input	Cursor name.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLSMALLINT	<i>cbCursor</i>	Input	Length of contents of <i>szCursor</i> argument.

### Usage

DB2 UDB CLI always generates and uses an internally generated cursor name when a SELECT statement is prepared or processed directly. SQLSetCursorName() allows an application-defined cursor name to be used in an SQL statement (a Positioned UPDATE or DELETE). DB2 UDB CLI maps this name to an internal name. SQLSetCursorName() must be called before an internal name is generated. The name remains associated with the statement handle, until the handle is dropped. The name also remains after the transaction has ended, but at this point SQLSetCursorName() can be called to set a different name for this statement handle.

Cursor names must follow the following 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.)
- Because 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 (\_).
- Unless the input cursor name is enclosed in double quotation marks, all leading and trailing blanks from the input cursor name string is removed.

### Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 151. *SQLSetCursorName* SQLSTATEs

SQLSTATE	Description	Explanation
34000	Cursor name that is not valid	The cursor name specified by the argument <i>szCursor</i> is not valid. The cursor name either begins with "SQLCUR" or "SQL_CUR" or violates either the driver or the data source 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 <i>szCursor</i> exists.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	<i>szCursor</i> is a null pointer.  The argument <i>cbCursor</i> is less than 1, but not equal to SQL_NTS.
HY010	Function sequence error	The statement handle is not in allocated state.  SQLPrepare() or SQLExecDirect() is called before SQLSetCursorName().
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## References

- "SQLGetCursorName - Get cursor name" on page 111

## SQLSetDescField - Set a descriptor field

### Purpose

SQLSetDescField() sets a field in a descriptor. SQLSetDescField() is a more extensible alternative to the SQLSetDescRec() function.

### Syntax

```
SQLRETURN SQLSetDescField (SQLHDESC      hdesc,  
                           SQLSMALLINT   irec,  
                           SQLSMALLINT   fDescType,  
                           SQLPOINTER    rgbDesc,  
                           SQLINTEGER    bLen);
```

### Function arguments

Table 152. *SQLSetDescField* arguments

Data type	Argument	Use	Description
SQLHDESC	<i>hdesc</i>	Input	Descriptor handle.
SQLSMALLINT	<i>irec</i>	Input	Record number from which the specified field is to be retrieved.
SQLSMALLINT	<i>fDescType</i>	Input	See Table 153 on page 194.
SQLPOINTER	<i>rgbDesc</i>	Input	Pointer to buffer.

Table 152. *SQLSetDescField* arguments (continued)

Data type	Argument	Use	Description
SQLINTEGER	<i>bLen</i>	Input	Length of descriptor buffer ( <i>rgbDesc</i> ).

Table 153. *fDescType* descriptor types

Descriptor	Type	Description
SQL_DESC_COUNT	SMALLINT	Set the number of records in the descriptor. <i>irec</i> is ignored.
SQL_DESC_DATA_PTR	SQLPOINTER	Set the data pointer field for <i>irec</i> .
SQL_DESC_DATETIME_INTERVAL_CODE	SMALLINT	Set the interval code for records with a type of SQL_DATETIME
SQL_DESC_INDICATOR_PTR	SQLPOINTER	Set the indicator pointer field for <i>irec</i> .
SQL_DESC_LENGTH_PTR	SQLPOINTER	Set the length pointer field for <i>irec</i> .
SQL_DESC_LENGTH	INTEGER	Set the length field of <i>irec</i> .
SQL_DESC_PRECISION	SMALLINT	Set the precision field of <i>irec</i> .
SQL_DESC_SCALE	SMALLINT	Set the scale field of <i>irec</i> .
SQL_DESC_TYPE	SMALLINT	Set the type field of <i>irec</i> .

## Usage

Instead of requiring an entire set of arguments like `SQLSetDescRec()`, `SQLSetDescField()` specifies which attribute you want to set for a specific descriptor record.

Although `SQLSetDescField()` allows for future extensions, it requires more calls to set the same information than `SQLSetDescRec()` for each descriptor record.

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 154. *SQLGetDescField* SQLSTATEs

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	The value specified for the argument <i>fDescType</i> or <i>irec</i> is not valid.  The argument <i>rgbValue</i> is a null pointer.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.



## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLPrepare - Prepare a statement” on page 164

## SQLSetDescRec - Set a descriptor record

### Purpose

SQLSetDescRec() sets all the attributes for a descriptor record. SQLSetDescRec() is a more concise alternative to the SQLDescField() function.

### Syntax

```
SQLRETURN SQLSetDescRec (SQLHDESC      hdesc,
                        SQLSMALLINT    irec,
                        SQLSMALLINT    type,
                        SQLSMALLINT    subtype,
                        SQLINTEGER      length,
                        SQLSMALLINT    prec,
                        SQLSMALLINT    scale,
                        SQLPOINTER      data,
                        SQLINTEGER      *sLen,
                        SQLINTEGER      *indic);
```

### Function arguments

Table 155. SQLSetDescRec arguments

Data type	Argument	Use	Description
SQLDESC	<i>hdesc</i>	Input	Descriptor handle.
SQLINTEGER *	<i>indic</i>	Input (deferred)	INDICATOR_PTR field for the record.
SQLINTEGER *	<i>sLen</i>	Input (deferred)	LENGTH_PTR field for the record.
SQLINTEGER	<i>length</i>	Input	LENGTH field for the record.
SQLPOINTER	<i>data</i>	Input (deferred)	DATA_PTR field for the record.
SQLSMALLINT	<i>irec</i>	Input	Record number within the descriptor.
SQLSMALLINT	<i>prec</i>	Input	PRECISION field for the record.
SQLSMALLINT	<i>scale</i>	Input	SCALE field for the record.
SQLSMALLINT	<i>subtype</i>	Input	DATETIME_INTERVAL_CODE field for records whose TYPE is SQL_DATETIME.
SQLSMALLINT	<i>type</i>	Input	TYPE field for the record.

### Usage

Calling SQLSetDescRec() sets all the fields in a descriptor record in one call.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO

- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 156. SQLSetDescRec SQLSTATEs

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	The value specified for the argument <i>irec</i> is less than 1.  A value that is not valid for another argument is specified.
HY016	Descriptor that is not valid	The descriptor handle referred to an implementation row descriptor.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## References

- “SQLBindCol - Bind a column to an application variable” on page 29
- “SQLDescribeCol - Describe column attributes” on page 68
- “SQLExecDirect - Execute a statement directly” on page 81
- “SQLExecute - Execute a statement” on page 83
- “SQLPrepare - Prepare a statement” on page 164

## SQLSetEnvAttr - Set environment attribute

### Purpose

| SQLSetEnvAttr() sets an environment attribute for the current environment. An environment attribute cannot be set if a connection handle has been allocated. In order for the attribute to apply to the entire CLI environment, the environment attributes must be in place before this initial connection is made. An HY010 error code is returned otherwise.

### Syntax

```
SQLRETURN SQLSetEnvAttr (SQLHENV      henv,
                        SQLINTEGER    Attribute,
                        SQLPOINTER    Value,
                        SQLINTEGER    StringLength);
```

### Function arguments

| Table 157. SQLSetEnvAttr arguments

Data type	Argument	Use	Description
SQLHENV	<i>henv</i>	Input	Environment handle.
SQLINTEGER	<i>Attribute</i>	Input	Environment attribute to set. Refer to Table 158 on page 197 for more information.
SQLINTEGER	<i>StringLength</i>	Input	Length of <i>Value</i> in bytes if the attribute value is a character string; if <i>Attribute</i> does not denote a string, then DB2 UDB CLI ignores <i>StringLength</i> .
SQLPOINTER	<i>pValue</i>	Input	Appropriate value for <i>Attribute</i> .

## Usage

Table 158. Environment attributes

Attribute	Contents
SQL_ATTR_DATE_FMT	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_FMT_ISO – The International Organization for Standardization (ISO) date format yyyy-mm-dd is used. This is the default.</li> <li>• SQL_FMT_USA – The United States date format mm/dd/yyyy is used.</li> <li>• SQL_FMT_EUR – The European date format dd.mm.yyyy is used.</li> <li>• SQL_FMT_JIS – The Japanese Industrial Standard date format yyyy-mm-dd is used.</li> <li>• SQL_FMT_MDY – The date format mm/dd/yy is used.</li> <li>• SQL_FMT_DMY – The date format dd/mm/yy is used.</li> <li>• SQL_FMT_YMD – The date format yy/mm/dd is used.</li> <li>• SQL_FMT_JUL – The Julian date format yy/ddd is used.</li> <li>• SQL_FMT_JOB – The job default is used.</li> </ul>
SQL_ATTR_DATE_SEP	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_SEP_SLASH – A slash ( / ) is used as the date separator. This is the default.</li> <li>• SQL_SEP_DASH – A dash ( - ) is used as the date separator.</li> <li>• SQL_SEP_PERIOD – A period ( . ) is used as the date separator.</li> <li>• SQL_SEP_COMMA – A comma ( , ) is used as the date separator.</li> <li>• SQL_SEP_BLANK – A blank is used as the date separator.</li> <li>• SQL_SEP_JOB – The job default is used.</li> </ul>
SQL_ATTR_DECIMAL_SEP	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_SEP_PERIOD – A period ( . ) is used as the decimal separator. This is the default.</li> <li>• SQL_SEP_COMMA – A comma ( , ) is used as the decimal separator.</li> <li>• SQL_SEP_JOB – The job default is used.</li> </ul>
SQL_ATTR_DEFAULT_LIB	<p>A character value that indicates the default library that is used for resolving unqualified file references. This is not valid if the environment is using system naming mode.</p>

Table 158. Environment attributes (continued)

Attribute	Contents
SQL_ATTR_ENVHNDL_COUNTER	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_FALSE – DB2 CLI does not count the number of times the environment handle is allocated. Therefore, the first call to free the environment handle and all associated resources.</li> <li>SQL_TRUE – DB2 CLI keeps a counter of the number of times the environment handle is allocated. Each time the environment handle is freed, the counter is decremented. Only when the counter reaches zero does the DB2 CLI actually free the handle and all associated resources. This allows nested calls to programs using the CLI that allocate and free the CLI environment handle.</li> </ul>
SQL_ATTR_ESCAPE_CHAR	<p>A character value that indicates the escape character to be used when specifying a search pattern in either SQLColumns( ) or SQLTables( ).</p>
SQL_ATTR_FOR_FETCH_ONLY	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_TRUE – Cursors are read-only and cannot be used for positioned update or delete operations. This is the default.</li> <li>SQL_FALSE – Cursors can be used for positioned updates or delete operations.</li> </ul> <p>The attribute SQL_ATTR_FOR_FETCH_ONLY can also be set for individual statements using SQLSetStmtAttr().</p>
SQL_ATTR_JOB_SORT_SEQUENCE	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_TRUE – DB2 UDB CLI uses the sort sequence that has been set for the job.</li> <li>SQL_FALSE – DB2 UDB CLI uses the default sort sequence, which is *HEX.</li> </ul>
SQL_ATTR_OUTPUT_NTS	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_TRUE – DB2 UDB CLI uses null termination to indicate the length of output character strings. This is the default.</li> <li>SQL_FALSE – DB2 UDB CLI does not use null termination.</li> </ul> <p>The CLI functions affected by this attribute are all functions called for the environment (and for any connections allocated under the environment) that have character string parameters.</p>
SQL_ATTR_REQUIRE_PROFILE	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_TRUE – If in server mode, then a profile and password are required when running SQLConnect() and SQLDriverConnect() functions.</li> <li>SQL_FALSE – If profile is omitted on the SQLConnect() or SQLDriverConnect() function, then connection is made using current user profile. This is the default.</li> </ul>

Table 158. Environment attributes (continued)

Attribute	Contents
SQL_ATTR_SERVER_MODE	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_FALSE – DB2 CLI processes the SQL statements of all connections within the same job. All changes compose a single transaction. This is the default mode of processing.</li> <li>• SQL_TRUE – DB2 CLI processes the SQL statements of each connection in a separate job. This allows multiple connections to the same data source, possibly with different user IDs for each connection. It also separates the changes made under each connection handle into its own transaction. This allows each connection handle to be committed or rolled back, without impacting pending changes made under other connection handles. See “Run DB2 UDB CLI in server mode” on page 246 for more information.</li> </ul>
SQL_ATTR_SYS_NAMING	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_TRUE – DB2 UDB CLI uses the iSeries system naming mode. Files are qualified using the slash (/) delimiter. Unqualified files are resolved using the library list for the job.</li> <li>• SQL_FALSE – DB2 UDB CLI uses the default naming mode, which is SQL naming. Files are qualified using the period (.) delimiter. Unqualified files are resolved using either the default library or the current user ID.</li> </ul>
SQL_ATTR_TIME_FMT	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_FMT_ISO – The International Organization for Standardization (ISO) time format hh.mm.ss is used. This is the default.</li> <li>• SQL_FMT_USA – The United States time format hh:mmxx is used, where xx is a.m. or p.m.</li> <li>• SQL_FMT_EUR – The European time format hh.mm.ss is used.</li> <li>• SQL_FMT_JIS – The Japanese Industrial Standard time format hh:mm:ss is used.</li> <li>• SQL_FMT_HMS – The hh:mm:ss format is used.</li> </ul>
SQL_ATTR_TIME_SEP	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>• SQL_SEP_COLON – A colon ( : ) is used as the time separator. This is the default.</li> <li>• SQL_SEP_PERIOD – A period ( . ) is used as the time separator.</li> <li>• SQL_SEP_COMMA – A comma ( , ) is used as the time separator.</li> <li>• SQL_SEP_BLANK – A blank is used as the time separator.</li> <li>• SQL_SEP_JOB – The job default is used.</li> </ul>

Table 158. Environment attributes (continued)

Attribute	Contents
SQL_ATTR_TRUNCATION_RTNC	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_TRUE – CLI returns SQL_SUCCESS_WITH_INFO in the SQLFetch() and SQLFetchScroll() return codes if truncation occurs.</li> <li>SQL_FALSE – CLI does not return SQL_SUCCESS_WITH_INFO in the SQLFetch() and SQLFetchScroll() return codes if truncation occurs. This is the default.</li> </ul>
SQL_ATTR_UTF8	<p>A 32-bit integer value:</p> <ul style="list-style-type: none"> <li>SQL_FALSE – Character data is treated as being in the default job coded character set identifier (CCSID). This is the default.</li> <li>SQL_TRUE – Character data is treated as being in the UTF-8 CCSID (1208).</li> </ul>

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 159. SQLSetEnvAttr SQLSTATEs

SQLSTATE	Description	Explanation
HY009	Parameter value that is not valid	<p>The specified <i>Attribute</i> is not supported by DB2 UDB CLI.</p> <p>Given specified <i>Attribute</i> value, the value specified for the argument <i>Value</i> is not supported.</p> <p>The argument <i>pValue</i> is a null pointer.</p>
HY010	Function sequence error	Connection handles are already allocated.

## SQLSetParam - Set parameter

### Purpose

SQLSetParam() has been deprecated and replaced by SQLBindParameter(). Although this version of DB2 UDB CLI continues to support SQLSetParam(), it is recommended that you begin using SQLBindParameter() in your DB2UDB CLI programs so that they conform to the latest standards.

SQLSetParam() associates (binds) an application variable to a parameter marker in an SQL statement. When the statement is processed, the contents of the bound variables are sent to the database server. This function is also used to specify any required data conversion.

### Syntax

```
SQLRETURN SQLSetParam (SQLHSTMT    hstmt,
                      SQLSMALLINT  ipar,
                      SQLSMALLINT  fctype,
```

```

SQLSMALLINT    fSqlType,
SQLINTEGER     cbParamDef,
SQLSMALLINT    ibScale,
SQLPOINTER     rgbValue,
SQLINTEGER     *pcbValue);

```

## SQLSetStmtAttr - Set a statement attribute

### Purpose

SQLSetStmtAttr() sets an attribute of a specific statement handle. To set an option for all statement handles associated with a connection handle, the application can call SQLSetConnectOption().

### Syntax

```

SQLRETURN SQLSetStmtAttr (SQLHSTMT    hstmt,
                          SQLINTEGER   fAttr,
                          SQLPOINTER   vParam,
                          SQLINTEGER   sLen);

```

### Function arguments

Table 160. SQLSetStmtAttr arguments

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLINTEGER	<i>fAttr</i>	Input	Attribute to set. Refer to Table 161 for the list of settable statement attributes.
SQLPOINTER	<i>vParam</i>	Input	Value associated with <i>fAttr</i> . <i>vParam</i> can be a 32-bit integer value or a character string.
SQLINTEGER	<i>sLen</i>	Input	Length of data if data is a character string; otherwise, unused.

### Usage

Statement options for an *hstmt* remain in effect until they are changed by another call to SQLSetStmtAttr() or the *hstmt* is dropped by calling SQLFreeStmt() with the SQL\_DROP option. Calling SQLFreeStmt() with the SQL\_CLOSE, SQL\_UNBIND, or SQL\_RESET\_PARAMS options does not reset the statement options.

The format of information set through *vParam* depends on the specified *fOption*. The format of each is noted in Table 161.

Table 161. Statement attributes

<i>fAttr</i>	Contents
SQL_ATTR_APP_PARAM_DESC	<i>vParam</i> must be a descriptor handle. The specified descriptor serves as the application parameter descriptor for later calls to SQLExecute() and SQLExecDirect() on the statement handle.
SQL_ATTR_APP_ROW_DESC	<i>vParam</i> must be a descriptor handle. The specified descriptor serves as the application row descriptor for later calls to SQLFetch() on the statement handle.

Table 161. Statement attributes (continued)

<i>fAttr</i>	Contents
SQL_ATTR_BIND_TYPE	<p>Specifies whether row-wise or column-wise binding is used.</p> <ul style="list-style-type: none"> <li>SQL_BIND_BY_ROW – Binding is row-wise. This is the default. When using row-wise binding for a multiple row fetch, all of the data for a row is returned in contiguous storage, followed by the data for the next row, and so on.</li> <li>SQL_BIND_BY_COLUMN – Binding is column-wise. When using column-wise binding for a multiple row fetch, all of the data for each column is returned in contiguous storage. The storage for each column need not be contiguous. A different address is provided by the user for each column in the result set, and it is the responsibility of the user to ensure that each address has space for all the data to be retrieved.</li> </ul>
SQL_ATTR_CURSOR_HOLD	<p>A 32-bit integer value that specifies if cursors opened for this statement handle should be held.</p> <ul style="list-style-type: none"> <li>SQL_FALSE – An open cursor for this statement handle is closed on a commit or rollback operation. This is the default.</li> <li>SQL_TRUE – An open cursor for this statement handle is not closed on a commit or rollback operation.</li> </ul>
SQL_ATTR_CURSOR_SCROLLABLE	<p>A 32-bit integer value that specifies if cursors opened for this statement handle should be scrollable.</p> <ul style="list-style-type: none"> <li>SQL_FALSE – Cursors are not scrollable, and <code>SQLFetchScroll()</code> cannot be used against them. This is the default.</li> <li>SQL_TRUE – Cursors are scrollable. <code>SQLFetchScroll()</code> can be used to retrieve data from these cursors.</li> </ul>
SQL_ATTR_CURSOR_SENSITIVITY	<p>A 32-bit integer value that specifies whether cursors opened for this statement handle make visible the changes made to the result set by another cursor. See <code>DECLARE CURSOR</code> for a more precise definition of the following options:</p> <ul style="list-style-type: none"> <li>SQL_UNSPECIFIED – Cursors on the statement handle might make visible none, some, or all such changes depending on the cursor type. This is the default.</li> <li>SQL_INSENSITIVE – All valid cursors on the statement handle show the result set without reflecting any changes made to it by any other cursor.</li> <li>SQL_SENSITIVE – All valid cursors on the statement handle make visible all changes made to a result by another cursor.</li> </ul>



Table 161. Statement attributes (continued)

<i>fAttr</i>	Contents
SQL_ATTR_CURSOR_TYPE	<p>A 32-bit integer value that specifies the behavior of cursors opened for this statement handle.</p> <ul style="list-style-type: none"> <li>• SQL_CURSOR_FORWARD_ONLY – Cursors are not scrollable, and the SQLFetchScroll() function cannot be used against them. This is the default.</li> <li>• SQL_CURSOR_DYNAMIC – Cursors are scrollable except for insensitive cursor sensitivity. The SQLFetchScroll() function can be used to retrieve data from these cursors.</li> <li>• SQL_CURSOR_STATIC – Cursors are scrollable except for sensitive cursor sensitivity. The SQLFetchScroll() function can be used to retrieve data from these cursors.</li> </ul>
SQL_ATTR_EXTENDED_COL_INFO	<p>A 32-bit integer value that specifies if cursors opened for this statement handle should provide extended column information.</p> <ul style="list-style-type: none"> <li>• SQL_FALSE – This statement handle cannot be used on the SQLColAttributes() function to retrieve extended column information. This is the default. Setting this attribute at the statement level overrides the connection level setting of the attribute.</li> <li>• SQL_TRUE – This statement handle can be used on the SQLColAttributes() function to retrieve extended column information, such as base table, base schema, base column, and label.</li> </ul>
SQL_ATTR_FOR_FETCH_ONLY	<p>A 32-bit integer value that specifies whether cursors opened for this statement handle should be read only:</p> <ul style="list-style-type: none"> <li>• SQL_TRUE – Cursors are read-only and cannot be used for positioned update or delete operations. This is the default unless SQL_ATTR_FOR_FETCH_ONLY environment has been set to SQL_FALSE.</li> <li>• SQL_FALSE – Cursors can be used for positioned update or delete operations.</li> </ul>
SQL_ATTR_FULL_OPEN	<p>A 32-bit integer value that specifies if cursors opened for this statement handle should be full open operations.</p> <ul style="list-style-type: none"> <li>• SQL_FALSE – Opening a cursor for this statement handle might use a cached cursor for performance reasons. This is the default.</li> <li>• SQL_TRUE – Opening a cursor for this statement handle always forces a full open operation of a new cursor.</li> </ul>

Table 161. Statement attributes (continued)

<i>fAttr</i>	Contents
SQL_ATTR_ROW_STATUS_PTR	<p>An output smallint pointer to specify an array of status values at SQLFetchScroll(). The number of elements must equal the number of rows in the row set (as defined by the SQL_ROWSET_SIZE attribute). A status value SQL_ROW_SUCCESS for each row fetched is returned.</p> <p>If the number of rows fetched is less than the number of elements in the status array (that is, less than the row set size), the remaining status elements are set to SQL_ROW_NOROW. The number of rows fetched is returned in the output pointer. This can be set by the SQLSetStmtAttr attribute SQL_ATTR_ROWS_FETCHED_PTR.</p> <p>DB2 UDB CLI cannot detect whether a row has been updated or deleted since the start of the fetch. Therefore, the following ODBC defined status values are not reported:</p> <ul style="list-style-type: none"> <li>• SQL_ROW_DELETED.</li> <li>• SQL_ROW_UPDATED.</li> </ul>
SQL_ATTR_ROWS_FETCHED_PTR	<p>An output integer pointer that contains the number of rows actually fetched by SQLFetchScroll(). If an error occurs during processing, the pointer points to the ordinal position of the row (in the row set) that precedes the row where the error occurred. If an error occurs retrieving the first row, the pointer points to the value 0.</p>
SQL_ATTR_ROWSET_SIZE	<p>A 32-bit integer value that specifies the number of rows in the row set. This is the number of rows returned by each call to SQLExtendedFetch(). The default value is 1.</p>

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 162. SQLStmtAttr SQLSTATEs

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
HY000	General error	An error occurred for which there is no specific SQLSTATE and for which no implementation defined SQLSTATE is defined. The error message returned by SQLError in the argument <i>szErrorMsg</i> describes the error and its cause.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.

Table 162. *SQLStmtAttr SQLSTATEs (continued)*

SQLSTATE	Description	Explanation
HY009	Argument value that is not valid	Given the specified <i>fAttr</i> value, a value that is not valid is specified for the argument <i>vParam</i> .  An <i>fAttr</i> value that is not valid is specified.  The argument <i>vParam</i> is a null pointer.
HY010	Function sequence error	The function is called out of sequence.
HYC00	Driver not capable	The driver or the data sources does not support the specified option.

#### Related reference

“SQLSetStmtOption - Set statement option”

“SQLFetchScroll - Fetch from a scrollable cursor” on page 92

## SQLSetStmtOption - Set statement option

### Purpose

- | SQLSetStmtOption() has been deprecated and replaced with SQLSetStmtAttr(). Although this version of
- | DB2 UDB CLI continues to support SQLSetStmtOption(), it is recommended that you begin using
- | SQLSetStmtAttr() in your DB2 UDB CLI programs so that they conform to the latest standards.

SQLSetStmtOption() sets an attribute of a specific statement handle. To set an option for all statement handles associated with a connection handle, the application can call SQLSetConnectAttr() (See “SQLSetConnectAttr - Set a connection attribute” on page 183 for additional details).

### Syntax

```
SQLRETURN SQLSetStmtOption (SQLHSTMT      hstmt,
                             SQLSMALLINT   fOption,
                             SQLPOINTER    vParam);
```

### Function arguments

Table 163. *SQLSetStmtOption arguments*

Data type	Argument	Use	Description
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLPOINTER	<i>vParam</i>	Input	Value associated with <i>fOption</i> . <i>vParam</i> can be a pointer to a 32-bit integer value or a character string.
SQLSMALLINT	<i>fOption</i>	Input	Option to set. Refer to Table 161 on page 201 for the list of settable statement options.

### Usage

- | The SQLSetStmtOption() provides many of the same attribute functions as SQLSetStmtAttr() prior to
- | V5R3. However, it has since been deprecated, and support for all new attribute functions has gone into
- | SQLSetStmtAttr(). Users should migrate to the nondeprecated interface.

Statement options for an *hstmt* remain in effect until they are changed by another call to SQLSetStmtOption() or the *hstmt* is dropped by calling SQLFreeStmt() with the SQL\_DROP option. Calling SQLFreeStmt() with the SQL\_CLOSE, SQL\_UNBIND, or SQL\_RESET\_PARAMS options does not reset statement options.

The format of information set through *vParam* depends on the specified *fOption*. The format of each is noted in Table 161 on page 201.

Refer to Table 161 on page 201 for the proper statement options.

**Note:** Because the `SQLSetStmtOption()` function has been deprecated, not all the options listed in the table are supported."

## Return codes

- `SQL_SUCCESS`
- `SQL_ERROR`
- `SQL_INVALID_HANDLE`

## Diagnostics

Table 164. *SQLStmtOption SQLSTATEs*

SQLSTATE	Description	Explanation
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
HY000	General error	An error occurred for which there is no specific SQLSTATE and for which no implementation defined SQLSTATE is defined. The error message returned by <code>SQLError</code> in the argument <i>szErrorMsg</i> describes the error and its cause.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument value that is not valid	Given the specified <i>fOption</i> value, a value that is not valid is specified for the argument <i>vParam</i> .  A <i>fOption</i> that is not valid value is specified.  The argument <i>szSchemaName</i> or <i>szTableName</i> is a null pointer.
HY010	Function sequence error	The function is called out of sequence.
HYC00	Driver not capable	The driver or the data sources does not support the specified option.

### Related reference

"`SQLSetStmtAttr` - Set a statement attribute" on page 201

"`SQLSetConnectAttr` - Set a connection attribute" on page 183

## SQLSpecialColumns - Get special (row identifier) columns

### Purpose

`SQLSpecialColumns()` returns unique row identifier information (primary key or unique index) for a table. For example, unique index or primary key information. 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 `SELECT`-statement.

### Syntax

```
SQLRETURN SQLSpecialColumns (SQLHSTMT      hstmt,
                             SQLSMALLINT   fColType,
                             SQLCHAR       *szCatalogName,
                             SQLSMALLINT   cbCatalogName,
                             SQLCHAR       *szSchemaName,
                             SQLSMALLINT   cbSchemaName,
                             SQLCHAR       *szTableName,
```

```

SQLSMALLINT  cbTableName,
SQLSMALLINT  fScope,
SQLSMALLINT  fNullable);

```

## Function arguments

Table 165. *SQLSpecialColumns* arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szCatalogName</i>	Input	Catalog qualifier of a three-part table name. This must be a null pointer or a zero length string.
SQLCHAR *	<i>szSchemaName</i>	Input	Schema qualifier of the specified table.
SQLCHAR *	<i>szTableName</i>	Input	Table name.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLSMALLINT	<i>cbCatalogName</i>	Input	Length of <i>szCatalogName</i> . This must be a set to 0.
SQLSMALLINT	<i>cbSchemaName</i>	Input	Length of <i>szSchemaName</i> .
SQLSMALLINT	<i>cbTableName</i>	Input	Length of <i>cbTableName</i> .
SQLSMALLINT	<i>fColType</i>	Input	Reserved for future use to support additional types of special columns.  This data type is currently ignored.
SQLSMALLINT	<i>fNullable</i>	Input	Determines whether to return special columns that can have a NULL value.  Must be one of the following values: <ul style="list-style-type: none"> <li>• <code>SQL_NO_NULLS</code> The row identifier column set returned cannot have any NULL values.</li> <li>• <code>SQL_NULLABLE</code> The row identifier column set returned can include columns where NULL values are permitted.</li> </ul>
SQLSMALLINT	<i>fScope</i>	Input	Minimum required duration for which the unique row identifier is valid.  <i>fScope</i> must be one of the following values: <ul style="list-style-type: none"> <li>• <code>SQL_SCOPE_CURROW</code> - The row identifier is guaranteed to be valid only while positioned on that row. A later reselect using the same row identifier values might not return a row if the row is updated or deleted by another transaction.</li> <li>• <code>SQL_SCOPE_TRANSACTION</code> - The row identifier is guaranteed to be valid for the duration of the current transaction.</li> <li>• <code>SQL_SCOPE_SESSION</code> - The row identifier is guaranteed to be valid for the duration of the connection.</li> </ul> The duration over which a row identifier value is guaranteed to be valid depends on the current transaction isolation level. For information and scenarios involving isolation levels, refer to the IBM DB2 SQL Reference.

## 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 UDB CLI returns the *best* set of row identifier columns based on its internal criterion.

If there is no column set that 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. The result set returned by `SQLSpecialColumns()` has the following columns in the following order:

Table 166. Columns returned by `SQLSpecialColumns`

Column name	Data type	Description
SCOPE	SMALLINT not NULL	Actual scope of the rowid. Contains one of the following values: <ul style="list-style-type: none"><li>• SQL_SCOPE_CURROW</li><li>• SQL_SCOPE_TRANSACTION</li><li>• SQL_SCOPE_SESSION</li></ul> Refer to <i>fScope</i> in Table 165 on page 207 for a description of each value.
COLUMN_NAME	VARCHAR(128) not NULL	Name of the row identifier column.
DATA_TYPE	SMALLINT not NULL	SQL data type of the column.
TYPE_NAME	VARCHAR(128) not NULL	Database Management System (DBMS) character string represented of the name associated with DATA_TYPE column value.
LENGTH_PRECISION	INTEGER	The precision of the column. NULL is returned for data types where precision is not applicable.
BUFFER_LENGTH	INTEGER	The length, in bytes, of the data returned in the default C type. For CHAR data types, this is the same as the value in the LENGTH_PRECISION column.
SCALE	SMALLINT	The scale of the column. NULL is returned for data types where scale is not applicable.
PSEUDO_COLUMN	SMALLINT	Indicates whether the column is a pseudo-column; DB2 UDB CLI only returns: <ul style="list-style-type: none"><li>• SQL_PC_NOT_PSEUDO</li></ul>

## Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 167. *SQLSpecialColumns SQLSTATES*

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	Cursor related information is requested, but no cursor is open.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument length that is not valid	The value of one of the length arguments is less than 0, but not equal to SQL_NTS.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYC00	Driver not capable	The data source does not support the <i>catalog</i> portion (first part) of a three-part table name.

## SQLStatistics - Get index and statistics information for a base table

### Purpose

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 fetch a result set generated by a SELECT-statement.

### Syntax

```
SQLRETURN SQLStatistics (SQLHSTMT      hstmt,
                        SQLCHAR        *szCatalogName,
                        SQLSMALLINT    cbCatalogName,
                        SQLCHAR        *szSchemaName,
                        SQLSMALLINT    cbSchemaName,
                        SQLCHAR        *szTableName,
                        SQLSMALLINT    cbTableName,
                        SQLSMALLINT    fUnique,
                        SQLSMALLINT    fAccuracy);
```

### Function arguments

Table 168. *SQLStatistics arguments*

Data type	Argument	Use	Description
SQLCHAR *	<i>szCatalogName</i>	Input	Catalog qualifier of a three-part table name. This must be a null pointer or a zero length string.
SQLCHAR *	<i>szSchemaName</i>	Input	Schema qualifier of the specified table.
SQLCHAR *	<i>szTableName</i>	Input	Table name.
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLSMALLINT	<i>cbCatalogName</i>	Input	Length of <i>cbCatalogName</i> . This must be set to 0.
SQLSMALLINT	<i>cbSchemaName</i>	Input	Length of <i>szSchemaName</i> .
SQLSMALLINT	<i>cbTableName</i>	Input	Length of <i>cbTableName</i> .
SQLSMALLINT	<i>fAccuracy</i>	Input	Not currently used, must be set to 0.

Table 168. SQLStatistics arguments (continued)

Data type	Argument	Use	Description
SQLSMALLINT	<i>fUnique</i>	Input	Type of index information to return: <ul style="list-style-type: none"> <li>• SQL_INDEX_UNIQUE Only unique indexes are returned.</li> <li>• SQL_INDEX_ALL All indexes are returned.</li> </ul>

## Usage

SQLStatistics() returns the following types of information:

- Statistics information for the table (if available):
  - When the TYPE column in the following table 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 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 the following table in the order shown; the rows in the result set are ordered by NON\_UNIQUE, TYPE, INDEX\_QUALIFIER, INDEX\_QUALIFIER, INDEX\_NAME and ORDINAL\_POSITION.

Table 169. Columns returned by SQLStatistics

Column name	Data type	Description
TABLE_CAT	VARCHAR(128)	The name of the catalog containing TABLE_SCHEM. This is set to NULL.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing TABLE_NAME.
TABLE_NAME	VARCHAR(128) not NULL	Name of the table.
NON_UNIQUE	SMALLINT	Indicates whether the index prohibits duplicate values: <ul style="list-style-type: none"> <li>• TRUE if the index allows duplicate values.</li> <li>• FALSE if the index values must be unique.</li> <li>• NULL is returned if the TYPE column indicates that this row is SQL_TABLE_STAT (statistics information about the table itself).</li> </ul>
INDEX_QUALIFIER	VARCHAR(128)	The identifier used to qualify the index name. This is NULL if the TYPE column indicates SQL_TABLE_STAT.
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.



Table 169. Columns returned by SQLStatistics (continued)

Column name	Data type	Description
TYPE	SMALLINT not NULL	<p>Indicates the type of information contained in this row of the result set:</p> <ul style="list-style-type: none"> <li>• SQL_TABLE_STAT Indicates this row contains statistics information on the table itself.</li> <li>• SQL_INDEX_CLUSTERED Indicates this row contains information about an index, and the index type is a clustered index.</li> <li>• SQL_INDEX_HASHED Indicates this row contains information about an index, and the index type is a hashed index.</li> <li>• SQL_INDEX_OTHER Indicates this row contains information about an index, and the index type is other than clustered or hashed.</li> </ul> <p><b>Note:</b> Currently, SQL_INDEX_OTHER is the only possible type.</p>
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_NAME	VARCHAR(128)	Name of the column in the index.
COLLATION	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.
CARDINALITY	INTEGER	<ul style="list-style-type: none"> <li>• If the TYPE column contains the value SQL_TABLE_STAT, this column contains the number of rows in the table.</li> <li>• If the TYPE column value is not SQL_TABLE_STAT, this column contains the number of unique values in the index.</li> <li>• A NULL value is returned if information is not available from the Database Management System (DBMS).</li> </ul>

Table 169. Columns returned by SQLStatistics (continued)

Column name	Data type	Description
PAGES	INTEGER	<ul style="list-style-type: none"> <li>• If the TYPE column contains the value SQL_TABLE_STAT, this column contains the number of pages used to store the table.</li> <li>• If the TYPE column value is not SQL_TABLE_STAT, this column contains the number of pages used to store the indexes.</li> <li>• A NULL value is returned if information is not available from the DBMS.</li> </ul>

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 COLLATION are set to NULL. If the CARDINALITY or PAGES information cannot be determined, then NULL is returned for those columns.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 170. SQLStatistics SQLSTATES

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	Cursor related information is requested, but no cursor is open.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal to SQL_NTS.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYC00	Driver not capable	The catalog part (the first part) of a three-part table name is not supported by the data source.

## SQLTablePrivileges – Get privileges associated with a table

### Purpose

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.

## Syntax

```
SQLRETURN SQLTablePrivileges (SQLHSTMT      StatementHandle,
                               SQLCHAR       *CatalogName,
                               SQLSMALLINT   NameLength1,
                               SQLCHAR       *SchemaName,
                               SQLSMALLINT   NameLength2,
                               SQLCHAR       *TableName,
                               SQLSMALLINT   NameLength3);
```

## Function arguments

Table 171. SQLTablePrivileges arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>SchemaName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by schema name.
SQLCHAR *	<i>szTableQualifier</i>	Input	Catalog qualifier of a 3 part table name. This must be a null pointer or a zero length string.
SQLCHAR *	<i>TableName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by table name.
SQLHSTMT	<i>StatementHandle</i>	Input	Statement handle.
SQLSMALLINT	<i>cbTableQualifier</i>	Input	Length of <i>CatalogName</i> . This must be set to 0.
SQLSMALLINT	<i>NameLength2</i>	Input	Length of <i>SchemaName</i> .
SQLSMALLINT	<i>NameLength3</i>	Input	Length of <i>TableName</i> .

## 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.

Because 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 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. Because DB2 names are less than 128, the application can choose to always set aside 128 characters (plus the null-terminator) for the output buffer, or alternatively, call SQLGetInfo() with SQL\_MAX\_CATALOG\_NAME\_LEN, SQL\_MAX\_SCHEMA\_NAME\_LEN, SQL\_MAX\_TABLE\_NAME\_LEN, and SQL\_MAX\_COLUMN\_NAME\_LEN. The SQL\_MAX\_CATALOG\_NAME\_LEN value determines the actual length of the TABLE\_CAT supported by the connected DBMS. The SQL\_MAX\_SCHEMA\_NAME\_LEN value determines the actual length of the TABLE\_SCHEM supported by the connected Database Management System (DBMS). The SQL\_MAX\_TABLE\_NAME\_LEN value determines the actual length of the TABLE\_NAME supported by the connected DBMS. The SQL\_MAX\_COLUMN\_NAME\_LEN value determines the actual length of the COLUMN\_NAME supported by the connected DBMS.

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

Table 172. Columns returned by SQLTablePrivileges

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Authorization ID of the user to whom the privilege is granted.
GRANTOR	VARCHAR(128)	Authorization ID of the user who granted the privilege.
IS_GRANTABLE	VARCHAR(3)	Indicates whether the grantee is permitted to grant the privilege to other users.  This can be "YES", "NO" or "NULL".
PRIVILEGE	VARCHAR(128)	The table privilege. This can be one of the following strings: <ul style="list-style-type: none"> <li>• ALTER</li> <li>• CONTROL</li> <li>• INDEX</li> <li>• DELETE</li> <li>• INSERT</li> <li>• REFERENCES</li> <li>• SELECT</li> <li>• UPDATE</li> </ul>
TABLE_CAT	VARCHAR(128)	This is always null.
TABLE_NAME	VARCHAR(128) not NULL	The name of the table.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing TABLE_NAME.

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 173. SQLTablePrivileges SQLSTATEs

SQLSTATE	Description	Explanation
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	String or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal SQL_NTS.
HY010	Function sequence error	There is an open cursor for this statement handle, or there is no connection for this statement handle.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.

## Restrictions

None.

## Example

```
/* From the CLI sample TBINFO.C */
/* ... */

/* call SQLTablePrivileges */
printf("\n    Call SQLTablePrivileges for:\n");
printf("        tbSchemaPattern = %s\n", tbSchemaPattern);
printf("        tbNamePattern = %s\n", tbNamePattern);
sqlrc = SQLTablePrivileges( hstmt, NULL, 0,
                           tbSchemaPattern, SQL_NTS,
                           tbNamePattern, SQL_NTS);
STMT_HANDLE_CHECK( hstmt, sqlrc);
```

## References

### SQLTables - Get table information

#### Purpose

SQLTables() returns a list of table names and associated information stored in the system catalogs 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 retrieve a result set generated by a SELECT statement.

#### Syntax

```
SQLRETURN SQLTables (SQLHSTMT    hstmt,
                    SQLCHAR      *szCatalogName,
                    SQLSMALLINT  cbCatalogName,
                    SQLCHAR      *szSchemaName,
                    SQLSMALLINT  cbSchemaName,
                    SQLCHAR      *szTableName,
                    SQLSMALLINT  cbTableName,
                    SQLCHAR      *szTableType,
                    SQLSMALLINT  cbTableType);
```

#### Function arguments

Table 174. SQLTables arguments

Data type	Argument	Use	Description
SQLCHAR *	<i>szCatalogName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set. <i>Catalog</i> is the first part of a three-part table name.  This must be a NULL pointer or a zero length string.
SQLCHAR *	<i>szSchemaName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by schema name.
SQLCHAR *	<i>szTableName</i>	Input	Buffer that might contain a <i>pattern-value</i> to qualify the result set by table name.

Table 174. *SQLTables* arguments (continued)

Data type	Argument	Use	Description
SQLCHAR *	<i>szTableType</i>	Input	<p>Buffer that might contain a <i>value list</i> to qualify the result set by table type.</p> <p>The value list is a list of values separated by commas for the types of interest. Valid table type identifiers might include: ALL, ALIAS, BASE TABLE, MATERIALIZED QUERY TABLE, SYSTEM TABLE, TABLE, VIEW. If <i>szTableType</i> 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.</p> <p>If SYSTEM TABLE is specified, then both system tables and system views (if there are any) are returned.</p> <p>The table types can be specified with or without quotation marks.</p>
SQLHSTMT	<i>hstmt</i>	Input	Statement handle.
SQLSMALLINT	<i>cbCatalogName</i>	Input	Length of <i>szCatalogName</i> . This must be set to 0.
SQLSMALLINT	<i>cbSchemaName</i>	Input	Length of <i>szSchemaName</i> .
SQLSMALLINT	<i>cbTableName</i>	Input	Length of <i>szTableName</i> .
SQLSMALLINT	<i>cbTableType</i>	Input	Size of <i>szTableType</i>

**Note:** The *szCatalogName*, *szSchemaName*, and *szTableName* arguments accept search patterns.

An escape character can be specified in conjunction with a wildcard character to allow that actual character to be used in the search pattern. The escape character is specified on the SQL\_ATTR\_ESCAPE\_CHAR environment attribute.

## Usage

Table information is returned in a result set where each table is represented by one row of the result set.

To support obtaining just a list of schemas, the following special semantics for the *szSchemaName* argument can be applied: if *szSchemaName* is a string containing a single percent (%) character, and *cbCatalogName*, *szTableName*, and *szTableType* are empty strings, then the result set contains a list of non-duplicate schemas in the data source.

The result set returned by *SQLTables()* contains the columns listed in the following table in the order given.

Table 175. Columns returned by *SQLTables*

Column name	Data type	Description
TABLE_CAT	VARCHAR(128)	The current server.
TABLE_SCHEM	VARCHAR(128)	The name of the schema containing TABLE_NAME.
TABLE_NAME	VARCHAR(128)	The name of the table, view, alias, or synonym.

Table 175. Columns returned by SQLTables (continued)

Column name	Data type	Description
TABLE_TYPE	VARCHAR(128)	Identifies the type given by the name in the TABLE_NAME column. It can have the string values ALIAS, BASE TABLE, MATERIALIZED QUERY TABLE, SYSTEM TABLE, TABLE, or VIEW.
REMARKS	VARCHAR(254)	Contains the descriptive information about the table.

### Return codes

- SQL\_SUCCESS
- SQL\_SUCCESS\_WITH\_INFO
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

### Diagnostics

Table 176. SQLTables SQLSTATES

SQLSTATE	Description	Explanation
24000	Cursor state that is not valid	Cursor-related information is requested, but no cursor is open.
40003 *	Statement completion unknown	The communication link between the CLI and the data source fails before the function completes processing.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY009	Argument or buffer length that is not valid	The value of one of the name length arguments is less than 0, but not equal to SQL_NTS.
HY021	Internal descriptor that is not valid	The internal descriptor cannot be addressed or allocated, or it contains a value that is not valid.
HYC00	Driver not capable	The catalog part (the first part) of a three-part table name is not supported by the data source.

## SQLTransact - Commit or roll back transaction

### Purpose

SQLTransact() commits or rolls back the current transaction in the connection.

All changes to the database performed on the connection since connect time or the previous call to SQLTransact() (whichever is the most recent) are committed or rolled back.

If a transaction is active on a connection, the application must call SQLTransact() before it can disconnect from the database.

### Syntax

```
SQLRETURN SQLTransact (SQLHENV      henv,
                      SQLHDBC      hdbc,
                      SQLSMALLINT  fType);
```

## Function arguments

Table 177. *SQLTransact arguments*

Data type	Argument	Use	Description
SQLHDBC	<i>hdbc</i>	Input	Database connection handle.  If <i>hdbc</i> is set to SQL_NULL_HDBC, then <i>henv</i> must contain the environment handle that the connection is associated with.
SQLHENV	<i>henv</i>	Input	Environment handle.  If <i>hdbc</i> is a valid connection handle, <i>henv</i> is ignored.
SQLSMALLINT	<i>fType</i>	Input	The wanted action for the transaction. The value for this argument must be one of: <ul style="list-style-type: none"><li>• SQL_COMMIT</li><li>• SQL_ROLLBACK</li><li>• SQL_COMMIT_HOLD</li><li>• SQL_ROLLBACK_HOLD</li></ul>

## Usage

Completing a transaction with SQL\_COMMIT or SQL\_ROLLBACK has the following effects:

- Statement handles are still valid after a call to SQLTransact().
- Cursor names, bound parameters, and column bindings survive transactions.
- Open cursors are closed, and any result sets that are pending retrieval are discarded.

Completing the transaction with SQL\_COMMIT\_HOLD or SQL\_ROLLBACK\_HOLD still commits or rolls back the database changes, but does not cause cursors to be closed.

If no transaction is currently active on the connection, calling SQLTransact() has no effect on the database server and returns SQL\_SUCCESS.

SQLTransact() might fail while executing the COMMIT or ROLLBACK due to a loss of connection. In this case the application might be unable to determine whether the COMMIT or ROLLBACK has been processed, and a database administrator's help might be required. Refer to the DBMS product information for more information about transaction logs and other transaction management tasks.

## Return codes

- SQL\_SUCCESS
- SQL\_ERROR
- SQL\_INVALID\_HANDLE

## Diagnostics

Table 178. *SQLTransact SQLSTATES*

SQLSTATE	Description	Explanation
08003	Connection not open	The <i>hdbc</i> is not in a connected state.



Table 178. SQLTransact SQLSTATEs (continued)

SQLSTATE	Description	Explanation
08007	Connection failure during transaction	The connection associated with the <i>hdbc</i> fails during the processing of the function during the processing of the function and it cannot be determined whether the requested COMMIT or ROLLBACK occurs before the failure.
58004	System error	Unrecoverable system error.
HY001	Memory allocation failure	The driver is unable to allocate memory required to support the processing or completion of the function.
HY012	Transaction operation state that is not valid	The value specified for the argument <i>fType</i> is neither SQL_COMMIT nor SQL_ROLLBACK.
HY013 *	Memory management problem	The driver is unable to access memory required to support the processing or completion of the function.

## Example

Refer to the example in “SQLFetch - Fetch next row” on page 87

### Related concepts

“Effects of calling SQLTransact() in a DB2 UDB CLI application” on page 14

## DB2 UDB CLI include file

The only include file used in DB2 UDB CLI is `sqlcli.h`.

```

/** START HEADER FILE SPECIFICATIONS *****/
/*
/* Header File Name: SQLCLI
/*
/* Product(s):
/* 5716-SS1
/* 5722-SS1
/*
/* (C)Copyright IBM Corp. 1995, 2003
/*
/* All rights reserved.
/* US Government Users Restricted Rights -
/* Use, duplication or disclosure restricted
/* by GSA ADP Schedule Contract with IBM Corp.
/*
/* Licensed Materials-Property of IBM
/* Descriptive Name: Structured Query Language (SQL) Call Level
/* Interface.
/*
/* Description: The SQL Call Level Interface provides access to
/* most SQL functions, without the need for a
/* precompiler.
/*
/* Header Files Included: SQLCLI
/*
/* Function Prototype List: SQLAllocConnect
/* SQLAllocEnv
/* SQLAllocHandle
/* SQLAllocStmt
/* SQLBindCol
/* SQLBindFileToCol
/* SQLBindFileToParam
/* SQLBindParam
/* SQLBindParameter

```

```

/*      SQLCancel      */
/*      SQLCloseCursor */
/*      SQLColAttributes */
/*      SQLColumnPrivileges */
/*      SQLColumns     */
/*      SQLConnect     */
/*      SQLCopyDesc    */
/*      SQLDataSources */
/*      SQLDescribeCol */
/*      SQLDescribeParam */
/*      SQLDisconnect  */
/*      SQLDriverConnect */
/*      SQLEndTran     */
/*      SQLERROR       */
/*      SQLExecDirect  */
/*      SQLExecute     */
/*      SQLExtendedFetch */
/*      SQLFetch       */
/*      SQLFetchScroll */
/*      SQLForeignKeys */
/*      SQLFreeConnect */
/*      SQLFreeEnv     */
/*      SQLFreeHandle  */
/*      SQLFreeStmt    */
/*      SQLGetCol      */
/*      SQLGetConnectOption */
/*      SQLGetCursorName */
/*      SQLGetConnectAttr */
/*      SQLGetData     */
/*      SQLGetDescField */
/*      SQLGetDescRec  */
/*      SQLGetDiagField */
/*      SQLGetDiagRec  */
/*      SQLGetEnvAttr  */
/*      SQLGetFunctions */
/*      SQLGetInfo     */
/*      SQLGetLength   */
/*      SQLGetPosition */
/*      SQLGetStmtAttr */
/*      SQLGetStmtOption */
/*      SQLGetSubString */
/*      SQLGetTypeInfo */
/*      SQLLanguages   */
/*      SQLMoreResults */
/*      SQLNativeSql   */
/*      SQLNextResult  */
/*      SQLNumParams   */
/*      SQLNumResultCols */
/*      SQLParamData   */
/*      SQLParamOptions */
/*      SQLPrepare     */
/*      SQLPrimaryKeys */
/*      SQLProcedureColumns */
/*      SQLProcedures  */
/*      SQLPutData     */
/*      SQLReleaseEnv  */
/*      SQLRowCount    */
/*      SQLSetConnectAttr */
/*      SQLSetConnectOption */
/*      SQLSetCursorName */
/*      SQLSetDescField */
/*      SQLSetDescRec  */
/*      SQLSetEnvAttr  */
/*      SQLSetParam    */
/*      SQLSetStmtAttr */
/*      SQLSetStmtOption */
/*      SQLSpecialColumns */

```

```

/*          SQLStartTran          */
/*          SQLStatistics         */
/*          SQLTablePrivileges   */
/*          SQLTables            */
/*          SQLTransact          */
/*          */
/* Change Activity:              */
/*          */
/* CFD List:                    */
/*          */
/* FLAG REASON      LEVEL DATE  PGMR      CHANGE DESCRIPTION  */
/* ----- */
/* $A0= D91823      3D60  941206 MEGERIAN  New Include           */
/* $A1= D94881      4D20  960816 MEGERIAN  V4R2M0 enhancements  */
/* $A2= D95600      4D30  970910 MEGERIAN  V4R3M0 enhancements  */
/* $A3= P3682850    4D40  981030 MEGERIAN  V4R4M0 enhancements  */
/* $A4= D97596      4D50  990326 LJAMESON  V4R5M0 enhancements  */
/* $A5= P9924900    5D10  000512 MEGERIAN  V5R1M0 enhancements  */
/* $C1= D98562      5D20  010107 MBAILEY  V5R2M0 enhancements  */
/* $C2= D9856201    5D20  010506 MBAILEY  More enhancements     */
/*          */
/* End CFD List.                */
/*          */
/* Additional notes about the Change Activity */
/* End Change Activity.         */
/**** END HEADER FILE SPECIFICATIONS *****/

```

```

#ifndef SQL_H_SQLCLI
#define SQL_H_SQLCLI          /* Permit duplicate Includes */

#if (__OS400_TGTVRM__>=510) /* @B1A*/
#pragma datamodel(P128)    /* @B1A*/
#endif                    /* @B1A*/

#ifdef __ILEC400__
#pragma checkout(suspend)
#pragma nomargins nosequence
#else
#pragma info(none)
#endif

#ifndef __SQL_EXTERN
#ifdef __ILEC400__
#define SQL_EXTERN extern
#else
#ifdef __cplusplus
#ifdef __TOS_OS400__
#define SQL_EXTERN extern "C nowiden"
#else
#define SQL_EXTERN extern "C"
#endif
#else
#define SQL_EXTERN extern
#endif /* __cplusplus */
#endif /* __ILEC_400__ */
#define __SQL_EXTERN
#endif

#ifdef __ILEC400__
#pragma argument (SQLAllocConnect , nowiden)
#pragma argument (SQLAllocEnv , nowiden)
#pragma argument (SQLAllocHandle , nowiden)
#pragma argument (SQLAllocStmt , nowiden)
#pragma argument (SQLBindCol , nowiden)
#pragma argument (SQLBindFileToCol , nowiden)
#pragma argument (SQLBindFileToParam , nowiden)
#pragma argument (SQLBindParam , nowiden)

```

```

#pragma argument (SQLBindParameter      , nowiden)
#pragma argument (SQLCancel             , nowiden)
#pragma argument (SQLCloseCursor        , nowiden)
#pragma argument (SQLColAttributes      , nowiden)
#pragma argument (SQLColumnPrivileges  , nowiden)
#pragma argument (SQLColumns            , nowiden)
#pragma argument (SQLConnect            , nowiden)
#pragma argument (SQLCopyDesc           , nowiden)
#pragma argument (SQLDataSources        , nowiden)
#pragma argument (SQLDescribeCol        , nowiden)
#pragma argument (SQLDescribeParam     , nowiden)
#pragma argument (SQLDisconnect         , nowiden)
#pragma argument (SQLDriverConnect      , nowiden)
#pragma argument (SQLEndTran           , nowiden)
#pragma argument (SQLError              , nowiden)
#pragma argument (SQLExecDirect         , nowiden)
#pragma argument (SQLExecute            , nowiden)
#pragma argument (SQLExtendedFetch     , nowiden)
#pragma argument (SQLFetch              , nowiden)
#pragma argument (SQLFetchScroll        , nowiden)
#pragma argument (SQLForeignKeys        , nowiden)
#pragma argument (SQLFreeConnect        , nowiden)
#pragma argument (SQLFreeEnv            , nowiden)
#pragma argument (SQLFreeHandle         , nowiden)
#pragma argument (SQLFreeStmt           , nowiden)
#pragma argument (SQLGetCol             , nowiden)
#pragma argument (SQLGetConnectOption   , nowiden)
#pragma argument (SQLGetCursorName      , nowiden)
#pragma argument (SQLGetConnectAttr     , nowiden)
#pragma argument (SQLGetData            , nowiden)
#pragma argument (SQLGetDescField       , nowiden)
#pragma argument (SQLGetDescRec         , nowiden)
#pragma argument (SQLGetDiagField       , nowiden)
#pragma argument (SQLGetDiagRec         , nowiden)
#pragma argument (SQLGetEnvAttr         , nowiden)
#pragma argument (SQLGetFunctions       , nowiden)
#pragma argument (SQLGetInfo            , nowiden)
#pragma argument (SQLGetLength          , nowiden)
#pragma argument (SQLGetPosition         , nowiden)
#pragma argument (SQLGetStmtAttr        , nowiden)
#pragma argument (SQLGetStmtOption      , nowiden)
#pragma argument (SQLGetSubString       , nowiden)
#pragma argument (SQLGetTypeInfo        , nowiden)
#pragma argument (SQLLanguages          , nowiden)
#pragma argument (SQLMoreResults        , nowiden)
#pragma argument (SQLNativeSql          , nowiden)
#pragma argument (SQLNextResult         , nowiden)
#pragma argument (SQLNumParams          , nowiden)
#pragma argument (SQLNumResultCols      , nowiden)
#pragma argument (SQLParamData          , nowiden)
#pragma argument (SQLParamOptions       , nowiden)
#pragma argument (SQLPrepare            , nowiden)
#pragma argument (SQLPrimaryKeys        , nowiden)
#pragma argument (SQLProcedureColumns   , nowiden)
#pragma argument (SQLProcedures         , nowiden)
#pragma argument (SQLPutData            , nowiden)
#pragma argument (SQLReleaseEnv         , nowiden)
#pragma argument (SQLRowCount            , nowiden)
#pragma argument (SQLSetConnectAttr     , nowiden)
#pragma argument (SQLSetConnectOption   , nowiden)
#pragma argument (SQLSetCursorName      , nowiden)
#pragma argument (SQLSetDescField       , nowiden)
#pragma argument (SQLSetDescRec         , nowiden)
#pragma argument (SQLSetEnvAttr         , nowiden)
#pragma argument (SQLSetParam           , nowiden)
#pragma argument (SQLSetStmtAttr        , nowiden)
#pragma argument (SQLSetStmtOption      , nowiden)

```

```

#pragma argument (SQLSpecialColumns , nowiden)
#pragma argument (SQLStartTran , nowiden)
#pragma argument (SQLStatistics , nowiden)
#pragma argument (SQLTablePrivileges , nowiden)
#pragma argument (SQLTables , nowiden)
#pragma argument (SQLTransact , nowiden)
#endif

/* generally useful constants */
#define SQL_FALSE 0
#define SQL_TRUE 1
#define SQL_NTS -3 /* NTS = Null Terminated String */
#define SQL_SQLSTATE_SIZE 5 /* size of SQLSTATE, not including
null terminating byte */

#define SQL_MAX_MESSAGE_LENGTH 512
#define SQL_MAX_OPTION_STRING_LENGTH 128

/* RETCODE values */
#define SQL_SUCCESS 0
#define SQL_SUCCESS_WITH_INFO 1
#define SQL_NO_DATA_FOUND 100
#define SQL_NEED_DATA 99
#define SQL_NO_DATA SQL_NO_DATA_FOUND
#define SQL_ERROR -1
#define SQL_INVALID_HANDLE -2
#define SQL_STILL_EXECUTING 2

/* SQLFreeStmt option values */
#define SQL_CLOSE 0
#define SQL_DROP 1
#define SQL_UNBIND 2
#define SQL_RESET_PARAMS 3

/* SQLSetParam defines */
#define SQL_C_DEFAULT 99

/* SQLEndTran option values */
#define SQL_COMMIT 0
#define SQL_ROLLBACK 1
#define SQL_COMMIT_HOLD 2
#define SQL_ROLLBACK_HOLD 3
#define SQL_SAVEPOINT_NAME_RELEASE 4
#define SQL_SAVEPOINT_NAME_ROLLBACK 5

/* SQLDriverConnect option values */
#define SQL_DRIVER_COMPLETE 1
#define SQL_DRIVER_COMPLETE_REQUIRED 1
#define SQL_DRIVER_NOPROMPT 1
#define SQL_DRIVER_PROMPT 0

/* Valid option codes for GetInfo procedure */
#define SQL_ACTIVE_CONNECTIONS 0
#define SQL_MAX_DRIVER_CONNECTIONS 0
#define SQL_MAX_CONCURRENT_ACTIVITIES 1
#define SQL_ACTIVE_STATEMENTS 1
#define SQL_PROCEDURES 2
#define SQL_DRIVER_NAME 6 /* @C1A*/
#define SQL_ODBC_API_CONFORMANCE 9 /* @C1A*/
#define SQL_ODBC_SQL_CONFORMANCE 10 /* @C1A*/
#define SQL_DBMS_NAME 17
#define SQL_DBMS_VER 18
#define SQL_DRIVER_VER 18
#define SQL_IDENTIFIER_CASE 28 /* @C1A*/
#define SQL_IDENTIFIER_QUOTE_CHAR 29 /* @C1A*/
#define SQL_MAX_COLUMN_NAME_LEN 30
#define SQL_MAX_CURSOR_NAME_LEN 31
#define SQL_MAX_OWNER_NAME_LEN 32

```

```

#define SQL_MAX_SCHEMA_NAME_LEN 33
#define SQL_MAX_TABLE_NAME_LEN 35
#define SQL_MAX_COLUMNS_IN_GROUP_BY 36
#define SQL_MAX_COLUMNS_IN_ORDER_BY 37
#define SQL_MAX_COLUMNS_IN_SELECT 38
#define SQL_MAX_COLUMNS_IN_TABLE 39
#define SQL_MAX_TABLES_IN_SELECT 40
#define SQL_COLUMN_ALIAS 41
#define SQL_DATA_SOURCE_NAME 42
#define SQL_DATASOURCE_NAME 42
#define SQL_DATABASE_NAME 42
#define SQL_MAX_COLUMNS_IN_INDEX 43
#define SQL_PROCEDURE_TERM 44 /* @C1A*/
#define SQL_QUALIFIER_TERM 45 /* @C1A*/
#define SQL_TXN_CAPABLE 46 /* @C1A*/
#define SQL_OWNER_TERM 47 /* @C1A*/
#define SQL_DATA_SOURCE_READ_ONLY 48 /* @C2A*/
#define SQL_DEFAULT_TXN_ISOLATION 49 /* @C2A*/
#define SQL_MULTIPLE_ACTIVE_TXN 55 /* @C2A*/
#define SQL_QUALIFIER_NAME_SEPARATOR 65 /* @C2A*/
#define SQL_CORRELATION_NAME 74 /* @C1A*/
#define SQL_NON_NULLABLE_COLUMNS 75 /* @C1A*/
#define SQL_DRIVER_ODBC_VER 77 /* @C1A*/
#define SQL_GROUP_BY 88 /* @C1A*/
#define SQL_ORDER_BY_COLUMNS_IN_SELECT 90 /* @C1A*/
#define SQL_OWNER_USAGE 91 /* @C1A*/
#define SQL_QUALIFIER_USAGE 92 /* @C1A*/
#define SQL_QUOTED_IDENTIFIER_CASE 93 /* @C1A*/
#define SQL_MAX_ROW_SIZE 104 /* @C1A*/
#define SQL_QUALIFIER_LOCATION 114 /* @C1A*/
#define SQL_MAX_CATALOG_NAME_LEN 115
#define SQL_MAX_STATEMENT_LEN 116
#define SQL_SEARCH_PATTERN_ESCAPE 117
#define SQL_OUTER_JOINS 118
#define SQL_LIKE_ESCAPE_CLAUSE 119
#define SQL_CATALOG_NAME 120
#define SQL_DESCRIBE_PARAMETER 121
#define SQL_STRING_FUNCTIONS 50
#define SQL_NUMERIC_FUNCTIONS 51
#define SQL_CONVERT_FUNCTIONS 52
#define SQL_TIMEDATE_FUNCTIONS 53
#define SQL_SQL92_PREDICATES 160
#define SQL_SQL92_VALUE_EXPRESSIONS 165
#define SQLAggregate_FUNCTIONS 169
#define SQL_SQL_CONFORMANCE 170
#define SQL_CONVERT_CHAR 171
#define SQL_CONVERT_NUMERIC 172
#define SQL_CONVERT_DECIMAL 173
#define SQL_CONVERT_INTEGER 174
#define SQL_CONVERT_SMALLINT 175
#define SQL_CONVERT_FLOAT 176
#define SQL_CONVERT_REAL 177
#define SQL_CONVERT_DOUBLE 178
#define SQL_CONVERT_VARCHAR 179
#define SQL_CONVERT_LONGVARCHAR 180
#define SQL_CONVERT_BINARY 181
#define SQL_CONVERT_VARBINARY 182
#define SQL_CONVERT_BIT 183
#define SQL_CONVERT_TINYINT 184
#define SQL_CONVERT_BIGINT 185
#define SQL_CONVERT_DATE 186
#define SQL_CONVERT_TIME 187
#define SQL_CONVERT_TIMESTAMP 188
#define SQL_CONVERT_LONGVARBINARY 189
#define SQL_CONVERT_INTERVAL_YEAR_MONTH 190
#define SQL_CONVERT_INTERVAL_DAY_TIME 191
#define SQL_CONVERT_WCHAR 192

```

```

#define SQL_CONVERT_WLONGVARCHAR 193
#define SQL_CONVERT_WVARCHAR 194
#define SQL_CONVERT_BLOB 195
#define SQL_CONVERT_CLOB 196
#define SQL_CONVERT_DBCLOB 197
#define SQL_CURSOR_COMMIT_BEHAVIOR 198
#define SQL_CURSOR_ROLLBACK_BEHAVIOR 199
#define SQL_POSITIONED_STATEMENTS 200
#define SQL_KEYWORDS 201
#define SQL_CONNECTION_JOB_NAME 202

/* Unsupported codes for SQLGetInfo */

#define SQL_LOCK_TYPES -1
#define SQL_POS_OPERATIONS -1
#define SQL_USER_NAME -1

/* Output values for cursor behavior */

#define SQL_CB_DELETE 1
#define SQL_CB_CLOSE 2
#define SQL_CB_PRESERVE 3

/* Aliased option codes (ODBC 3.0) @C1A*/
#define SQL_SCHEMA_TERM SQL_OWNER_TERM /* @C1A*/
#define SQL_SCHEMA_USAGE SQL_OWNER_USAGE /* @C1A*/
#define SQL_CATALOG_LOCATION SQL_QUALIFIER_LOCATION /*@C1A*/
#define SQL_CATALOG_TERM SQL_QUALIFIER_TERM /* @C1A*/
#define SQL_CATALOG_USAGE SQL_QUALIFIER_USAGE /* @C1A*/
#define SQL_CATALOG_NAME_SEPARATOR SQL_QUALIFIER_NAME_SEPARATOR /* @C2A*/

/*
 * Output values for SQL_ODBC_API_CONFORMANCE
 * info type in SQLGetInfo
 */
#define SQL_OAC_NONE 0 /* @C1A*/
#define SQL_OAC_LEVEL1 1 /* @C1A*/
#define SQL_OAC_LEVEL2 2 /* @C1A*/

/*
 * Output values for SQL_ODBC_SQL_CONFORMANCE
 * info type in SQLGetInfo
 */
#define SQL_OSC_MINIMUM 0 /* @C1A*/
#define SQL_OSC_CORE 1 /* @C1A*/
#define SQL_OSC_EXTENDED 2 /* @C1A*/

/*
 * Output values for SQL_QUALIFIER_USAGE
 * info type in SQLGetInfo
 */
#define SQL_QU_NOT_SUPPORTED 0x00000000 /* @C1A*/
#define SQL_QU_DML_STATEMENTS 0x00000001 /* @C1A*/
#define SQL_QU_PROCEDURE_INVOCATION 0x00000002 /* @C1A*/
#define SQL_QU_TABLE_DEFINITION 0x00000004 /* @C1A*/
#define SQL_QU_INDEX_DEFINITION 0x00000008 /* @C1A*/
#define SQL_QU_PRIVILEGE_DEFINITION 0x00000010 /* @C1A*/

/*
 * Output values for SQL_QUALIFIER_LOCATION
 * info type in SQLGetInfo
 */
#define SQL_QL_START 1 /* @C1A*/
#define SQL_QL_END 2 /* @C1A*/

```

```

/*
 * Output values for SQL_OWNER_USAGE
 * info type in SQLGetInfo
 */
#define SQL_OU_DML_STATEMENTS          0x00000001 /* @C1A*/
#define SQL_OU_PROCEDURE_INVOCATION    0x00000002 /* @C1A*/
#define SQL_OU_TABLE_DEFINITION        0x00000004 /* @C1A*/
#define SQL_OU_INDEX_DEFINITION        0x00000008 /* @C1A*/
#define SQL_OU_PRIVILEGE_DEFINITION    0x00000010 /* @C1A*/

/*
 * Output values for SQL_TXN_CAPABLE
 * info type in SQLGetInfo
 */
#define SQL_TC_NONE                    0 /* @C1A*/
#define SQL_TC_DML                     1 /* @C1A*/
#define SQL_TC_ALL                     2 /* @C1A*/
#define SQL_TC_DDL_COMMIT              3 /* @C1A*/
#define SQL_TC_DDL_IGNORE              4 /* @C1A*/

/*
 * Output values for SQL_DEFAULT_TXN_ISOLATION
 * info type in SQLGetInfo
 */
#define SQL_TXN_READ_UNCOMMITTED_MASK 0x00000001 /* @C2A*/
#define SQL_TXN_READ_COMMITTED_MASK   0x00000002 /* @C2A*/
#define SQL_TXN_REPEATABLE_READ_MASK  0x00000004 /* @C2A*/
#define SQL_TXN_SERIALIZABLE_MASK     0x00000008 /* @C2A*/

/*
 * Output values for SQL_STRING_FUNCTIONS
 * info type in SQLGetInfo
 */
#define SQL_FN_STR_CONCAT              0x00000001
#define SQL_FN_STR_UCASE               0x00000002
#define SQL_FN_STR_LCASE               0x00000004
#define SQL_FN_STR_SUBSTRING           0x00000008
#define SQL_FN_STR_LENGTH              0x00000010
#define SQL_FN_STR_POSITION            0x00000020
#define SQL_FN_STR_LTRIM               0x00000040
#define SQL_FN_STR_RTRIM               0x00000080

/*
 * Output values for SQL_POS_OPERATIONS
 * info type in SQLGetInfo (not currently supported)
 */
#define SQL_POS_POSITION               0x00000001
#define SQL_POS_REFRESH                0x00000002
#define SQL_POS_UPDATE                 0x00000004
#define SQL_POS_DELETE                 0x00000008
#define SQL_POS_ADD                     0x00000010

/*
 * Output values for SQL_NUMERIC_FUNCTIONS
 * info type in SQLGetInfo
 */
#define SQL_FN_NUM_ABS                 0x00000001
#define SQL_FN_NUM_ACOS                0x00000002
#define SQL_FN_NUM_ASIN                0x00000004
#define SQL_FN_NUM_ATAN                0x00000008
#define SQL_FN_NUM_ATAN2               0x00000010
#define SQL_FN_NUM_CEILING             0x00000020
#define SQL_FN_NUM_COS                 0x00000040
#define SQL_FN_NUM_COT                 0x00000080
#define SQL_FN_NUM_EXP                  0x00000100

```



```

#define SQL_FN_NUM_FLOOR          0x00000200
#define SQL_FN_NUM_LOG           0x00000400
#define SQL_FN_NUM_MOD           0x00000800
#define SQL_FN_NUM_SIGN         0x00001000
#define SQL_FN_NUM_SIN          0x00002000
#define SQL_FN_NUM_NUM_SQRT     0x00004000
#define SQL_FN_NUM_NUM_TAN     0x00008000
#define SQL_FN_NUM_NUM_PI       0x00010000
#define SQL_FN_NUM_NUM_RAND     0x00020000
#define SQL_FN_NUM_NUM_DEGREES  0x00040000
#define SQL_FN_NUM_NUM_LOG10    0x00080000
#define SQL_FN_NUM_NUM_POWER    0x00100000
#define SQL_FN_NUM_NUM_RADIANS  0x00200000
#define SQL_FN_NUM_NUM_ROUND    0x00400000
#define SQL_FN_NUM_NUM_TRUNCATE 0x00800000

/* SQL_SQL92_VALUE_EXPRESSIONS bit masks */
#define SQL_SVE_CASE              0x00000001
#define SQL_SVE_CAST             0x00000002
#define SQL_SVE_COALESCE        0x00000004
#define SQL_SVE_NULLIF         0x00000008

/* SQL_SQL92_PREDICATES bit masks */
#define SQL_SP_EXISTS            0x00000001
#define SQL_SP_ISNOTNULL        0x00000002
#define SQL_SP_ISNULL          0x00000004
#define SQL_SP_MATCH_FULL      0x00000008
#define SQL_SP_MATCH_PARTIAL   0x00000010
#define SQL_SP_MATCH_UNIQUE_FULL 0x00000020
#define SQL_SP_MATCH_UNIQUE_PARTIAL 0x00000040
#define SQL_SP_OVERLAPS        0x00000080
#define SQL_SP_UNIQUE          0x00000100
#define SQL_SP_LIKE             0x00000200
#define SQL_SP_IN               0x00000400
#define SQL_SP_BETWEEN         0x00000800
#define SQL_SP_COMPARISON      0x00001000
#define SQL_SP_QUANTIFIED_COMPARISON 0x00002000

/* SQLAggregate_FUNCTIONS bit masks */
#define SQL_AF_AVG              0x00000001
#define SQL_AF_COUNT           0x00000002
#define SQL_AF_MAX             0x00000004
#define SQL_AF_MIN             0x00000008
#define SQL_AF_SUM             0x00000010
#define SQL_AF_DISTINCT        0x00000020
#define SQL_AF_ALL             0x00000040

/* SQL_SQL_CONFORMANCE bit masks */
#define SQL_SC_SQL92_ENTRY      0x00000001
#define SQL_SC_FIPS127_2_TRANSITIONAL 0x00000002
#define SQL_SC_SQL92_INTERMEDIATE 0x00000004
#define SQL_SC_SQL92_FULL      0x00000008

/* SQL_CONVERT_FUNCTIONS functions */
#define SQL_FN_CVT_CONVERT      0x00000001
#define SQL_FN_CVT_CAST        0x00000002

/* SQL_POSITIONED_STATEMENTS bit masks */
#define SQL_PS_POSITIONED_DELETE 0x00000001
#define SQL_PS_POSITIONED_UPDATE 0x00000002
#define SQL_PS_SELECT_FOR_UPDATE 0x00000004

/* SQL supported conversion bit masks */
#define SQL_CVT_CHAR            0x00000001
#define SQL_CVT_NUMERIC         0x00000002
#define SQL_CVT_DECIMAL         0x00000004
#define SQL_CVT_INTEGER         0x00000008

```

```

#define SQL_CVT_SMALLINT          0x00000010
#define SQL_CVT_FLOAT             0x00000020
#define SQL_CVT_REAL              0x00000040
#define SQL_CVT_DOUBLE            0x00000080
#define SQL_CVT_VARCHAR           0x00000100
#define SQL_CVT_LONGVARCHAR       0x00000200
#define SQL_CVT_BINARY            0x00000400
#define SQL_CVT_VARBINARY         0x00000800
#define SQL_CVT_BIT               0x00001000
#define SQL_CVT_TINYINT           0x00002000
#define SQL_CVT_BIGINT            0x00004000
#define SQL_CVT_DATE              0x00008000
#define SQL_CVT_TIME              0x00010000
#define SQL_CVT_TIMESTAMP         0x00020000
#define SQL_CVT_LONGVARBINARY    0x00040000
#define SQL_CVT_INTERVAL_YEAR_MONTH 0x00080000
#define SQL_CVT_INTERVAL_DAY_TIME 0x00100000
#define SQL_CVT_WCHAR             0x00200000
#define SQL_CVT_WLONGVARCHAR      0x00400000
#define SQL_CVT_WVARCHAR          0x00800000
#define SQL_CVT_BLOB              0x01000000
#define SQL_CVT_CLOB              0x02000000
#define SQL_CVT_DBCLOB            0x04000000

/* SQL_TIMEDATE_FUNCTIONS bit masks */
#define SQL_FN_TD_NOW             0x00000001
#define SQL_FN_TD_CURDATE        0x00000002
#define SQL_FN_TD_DAYOFMONTH     0x00000004
#define SQL_FN_TD_DAYOFWEEK      0x00000008
#define SQL_FN_TD_DAYOFYEAR      0x00000010
#define SQL_FN_TD_MONTH          0x00000020
#define SQL_FN_TD_QUARTER        0x00000040
#define SQL_FN_TD_WEEK           0x00000080
#define SQL_FN_TD_YEAR           0x00000100
#define SQL_FN_TD_CURTIME        0x00000200
#define SQL_FN_TD_HOUR           0x00000400
#define SQL_FN_TD_MINUTE         0x00000800
#define SQL_FN_TD_SECOND         0x00001000
#define SQL_FN_TD_TIMESTAMPADD   0x00002000
#define SQL_FN_TD_TIMESTAMPDIFF  0x00004000
#define SQL_FN_TD_DAYNAME        0x00008000
#define SQL_FN_TD_MONTHNAME      0x00010000
#define SQL_FN_TD_CURRENT_DATE   0x00020000
#define SQL_FN_TD_CURRENT_TIME   0x00040000
#define SQL_FN_TD_CURRENT_TIMESTAMP 0x00080000
#define SQL_FN_TD_EXTRACT        0x00100000

/*
 * Output values for SQL_CORRELATION_NAME
 * info type in SQLGetInfo
 */
#define SQL_CN_NONE              0          /* @C1A*/
#define SQL_CN_DIFFERENT         1          /* @C1A*/
#define SQL_CN_ANY               2          /* @C1A*/

/*
 * Output values for SQL_IDENTIFIER_CASE
 * info type in SQLGetInfo
 */
#define SQL_IC_UPPER              1          /* @C1A*/
#define SQL_IC_LOWER              2          /* @C1A*/
#define SQL_IC_SENSITIVE          3          /* @C1A*/
#define SQL_IC_MIXED              4          /* @C1A*/

/*
 * Output values for SQL_NON_NULLABLE_COLUMNS
 * info type in SQLGetInfo

```

```

*/
#define SQL_NNC_NULL          0          /* @C1A*/
#define SQL_NNC_NON_NULL     1          /* @C1A*/

/*
 * Output values for SQL_GROUP_BY
 * info type in SQLGetInfo
 */
#define SQL_GB_NO_RELATION    0          /* @C1A*/
#define SQL_GB_NOT_SUPPORTED  1          /* @C1A*/
#define SQL_GB_GROUP_BY_EQUALS_SELECT 2 /* @C1A*/
#define SQL_GB_GROUP_BY_CONTAINS_SELECT 3 /* @C1A*/

/* Standard SQL data types */
#define SQL_CHAR              1
#define SQL_NUMERIC          2
#define SQL_DECIMAL          3
#define SQL_INTEGER          4
#define SQL_SMALLINT         5
#define SQL_FLOAT            6
#define SQL_REAL             7
#define SQL_DOUBLE           8
#define SQL_DATETIME        9
#define SQL_VARCHAR         12
#define SQL_BLOB            13
#define SQL_CLOB            14
#define SQL_DBCLLOB         15
#define SQL_DATALINK        16
#define SQL_WCHAR           17
#define SQL_WVARCHAR        18
#define SQL_BIGINT          19
#define SQL_BLOB_LOCATOR    20
#define SQL_CLOB_LOCATOR    21
#define SQL_DBCLLOB_LOCATOR 22
#define SQL_WLONGVARCHAR    SQL_WVARCHAR
#define SQL_LONGVARCHAR     SQL_VARCHAR
#define SQL_GRAPHIC         95
#define SQL_VARGRAPHIC     96
#define SQL_LONGVARGRAPHIC SQL_VARGRAPHIC
#define SQL_BINARY          97
#define SQL_VARBINARY       98
#define SQL_LONGVARBINARY  SQL_VARBINARY
#define SQL_DATE            91
#define SQL_TYPE_DATE       91
#define SQL_TIME            92
#define SQL_TYPE_TIME       92
#define SQL_TIMESTAMP       93
#define SQL_TYPE_TIMESTAMP  93
#define SQL_CODE_DATE       1
#define SQL_CODE_TIME       2
#define SQL_CODE_TIMESTAMP  3
#define SQL_ALL_TYPES       0

/* Handle types */
#define SQL_UNUSED          0
#define SQL_HANDLE_ENV      1
#define SQL_HANDLE_DBC      2
#define SQL_HANDLE_STMT     3
#define SQL_HANDLE_DESC     4
#define SQL_NULL_HANDLE     0

#define SQL_HANDLE_DBC_UNICODE 100

/*
 * NULL status defines; these are used in SQLColAttributes, SQLDescribeCol,
 * to describe the nullability of a column in a table.
 */

```

```

#define SQL_NO_NULLS          0
#define SQL_NULLABLE         1
#define SQL_NULLABLE_UNKNOWN 2

/* Special length values */
#define SQL_NO_TOTAL          0
#define SQL_NULL_DATA        -1
#define SQL_DATA_AT_EXEC     -2
#define SQL_BIGINT_PREC      19
#define SQL_INTEGER_PREC     10
#define SQL_SMALLINT_PREC    5

/* SQLColAttributes defines */
#define SQL_ATTR_READONLY     0
#define SQL_ATTR_WRITE       1
#define SQL_ATTR_READWRITE_UNKNOWN 2

/* Valid concurrency values */
#define SQL_CONCUR_LOCK      0
#define SQL_CONCUR_READ_ONLY 1
#define SQL_CONCUR_ROWVER    3
#define SQL_CONCUR_VALUES    4

/* Valid environment attributes */
#define SQL_ATTR_OUTPUT_NTS   10001
#define SQL_ATTR_SYS_NAMING   10002
#define SQL_ATTR_DEFAULT_LIB  10003
#define SQL_ATTR_SERVER_MODE  10004
#define SQL_ATTR_JOB_SORT_SEQUENCE 10005
#define SQL_ATTR_ENVHNDL_COUNTER 10009
#define SQL_ATTR_ESCAPE_CHAR  10010
#define SQL_ATTR_INCLUDE_NULL_IN_LEN 10031
#define SQL_ATTR_UTF8         10032
#define SQL_ATTR_SYSCAP       10033
#define SQL_ATTR_REQUIRE_PROFILE 10034
#define SQL_ATTR_UCS2         10035

/* Valid environment/connection attributes */
#define SQL_ATTR_EXTENDED_COL_INFO 10019
#define SQL_ATTR_DATE_FMT         10020
#define SQL_ATTR_DATE_SEP        10021
#define SQL_ATTR_TIME_FMT        10022
#define SQL_ATTR_TIME_SEP        10023
#define SQL_ATTR_DECIMAL_SEP     10024
#define SQL_ATTR_TXN_INFO        10025
#define SQL_ATTR_TXN_EXTERNAL    10026
#define SQL_ATTR_2ND_LEVEL_TEXT  10027
#define SQL_ATTR_SAVEPOINT_NAME  10028
#define SQL_ATTR_TRACE           10029
#define SQL_ATTR_MAX_PRECISION   10040
#define SQL_ATTR_MAX_SCALE       10041
#define SQL_ATTR_MIN_DIVIDE_SCALE 10042
#define SQL_ATTR_HEX_LITERALS    10043

/* Valid transaction info operations */
#define SQL_TXN_FIND            1
#define SQL_TXN_CREATE         2
#define SQL_TXN_CLEAR          3
#define SQL_TXN_END            4

/* Valid environment/connection values */
#define SQL_FMT_ISO             1
#define SQL_FMT_USA            2
#define SQL_FMT_EUR            3
#define SQL_FMT_JIS            4
#define SQL_FMT_MDY            5

```

```

#define SQL_FMT_DMY          6
#define SQL_FMT_YMD          7
#define SQL_FMT_JUL          8
#define SQL_FMT_HMS          9
#define SQL_FMT_JOB         10
#define SQL_SEP_SLASH        1
#define SQL_SEP_DASH         2
#define SQL_SEP_PERIOD       3
#define SQL_SEP_COMMA        4
#define SQL_SEP_BLANK        5
#define SQL_SEP_COLON        6
#define SQL_SEP_JOB          7
#define SQL_HEX_IS_CHAR      1
#define SQL_HEX_IS_BINARY    2

/* Valid values for type in GetCol */
#define SQL_DEFAULT          99
#define SQL_ARD_TYPE         -99

/* Valid values for UPDATE_RULE and DELETE_RULE in SQLForeignKeys */
#define SQL_CASCADE          1
#define SQL_RESTRICT         2
#define SQL_NO_ACTION        3
#define SQL_SET_NULL         4
#define SQL_SET_DEFAULT      5

/* Valid values for COLUMN_TYPE in SQLProcedureColumns */
#define SQL_PARAM_INPUT      1
#define SQL_PARAM_OUTPUT     2
#define SQL_PARAM_INPUT_OUTPUT 3

/* statement attributes */
#define SQL_ATTR_APP_ROW_DESC 10010
#define SQL_ATTR_APP_PARAM_DESC 10011
#define SQL_ATTR_IMP_ROW_DESC 10012
#define SQL_ATTR_IMP_PARAM_DESC 10013
#define SQL_ATTR_FOR_FETCH_ONLY 10014
#define SQL_ATTR_CONCURRENCY 10014
#define SQL_CONCURRENCY 10014
#define SQL_ATTR_CURSOR_SCROLLABLE 10015
#define SQL_ATTR_ROWSET_SIZE 10016
#define SQL_ROWSET_SIZE 10016
#define SQL_ATTR_ROW_ARRAY_SIZE 10016
#define SQL_ATTR_CURSOR_HOLD 10017
#define SQL_ATTR_FULL_OPEN 10018
#define SQL_ATTR_BIND_TYPE 10049
#define SQL_BIND_TYPE 10049
#define SQL_ATTR_CURSOR_TYPE 10050
#define SQL_CURSOR_TYPE 10050

/* values for setting statement attributes */
#define SQL_BIND_BY_ROW 0
#define SQL_BIND_BY_COLUMN 1
#define SQL_CURSOR_FORWARD_ONLY 0
#define SQL_CURSOR_STATIC 1
#define SQL_CURSOR_DYNAMIC 2
#define SQL_CURSOR_KEYSET_DRIVEN 3

/* Codes used in FetchScroll */
#define SQL_FETCH_NEXT 1
#define SQL_FETCH_FIRST 2
#define SQL_FETCH_LAST 3
#define SQL_FETCH_PRIOR 4
#define SQL_FETCH_ABSOLUTE 5
#define SQL_FETCH_RELATIVE 6

/* SQLColAttributes defines */

```

```

#define SQL_DESC_COUNT                1
#define SQL_DESC_TYPE                 2
#define SQL_DESC_LENGTH              3
#define SQL_DESC_LENGTH_PTR          4
#define SQL_DESC_PRECISION           5
#define SQL_DESC_SCALE               6
#define SQL_DESC_DATETIME_INTERVAL_CODE 7
#define SQL_DESC_NULLABLE            8
#define SQL_DESC_INDICATOR_PTR       9
#define SQL_DESC_DATA_PTR            10
#define SQL_DESC_NAME                11
#define SQL_DESC_UNNAMED              12
#define SQL_DESC_DISPLAY_SIZE        13
#define SQL_DESC_AUTO_INCREMENT      14
#define SQL_DESC_SEARCHABLE          15
#define SQL_DESC_UPDATABLE           16
#define SQL_DESC_BASE_COLUMN         17
#define SQL_DESC_BASE_TABLE          18
#define SQL_DESC_BASE_SCHEMA         19
#define SQL_DESC_LABEL                20
#define SQL_DESC_MONEY               21
#define SQL_DESC_ALLOC_TYPE          99
#define SQL_DESC_ALLOC_AUTO          1
#define SQL_DESC_ALLOC_USER          2

#define SQL_COLUMN_COUNT              1
#define SQL_COLUMN_TYPE               2
#define SQL_COLUMN_LENGTH             3
#define SQL_COLUMN_LENGTH_PTR         4
#define SQL_COLUMN_PRECISION          5
#define SQL_COLUMN_SCALE              6
#define SQL_COLUMN_DATETIME_INTERVAL_CODE 7
#define SQL_COLUMN_NULLABLE           8
#define SQL_COLUMN_INDICATOR_PTR      9
#define SQL_COLUMN_DATA_PTR           10
#define SQL_COLUMN_NAME               11
#define SQL_COLUMN_UNNAMED            12
#define SQL_COLUMN_DISPLAY_SIZE       13
#define SQL_COLUMN_AUTO_INCREMENT     14
#define SQL_COLUMN_SEARCHABLE         15
#define SQL_COLUMN_UPDATABLE          16
#define SQL_COLUMN_BASE_COLUMN        17
#define SQL_COLUMN_BASE_TABLE         18
#define SQL_COLUMN_BASE_SCHEMA        19
#define SQL_COLUMN_LABEL              20
#define SQL_COLUMN_MONEY               21
#define SQL_COLUMN_ALLOC_TYPE         99
#define SQL_COLUMN_ALLOC_AUTO         1
#define SQL_COLUMN_ALLOC_USER         2

/* Valid codes for SpecialColumns procedure */
#define SQL_SCOPE_CURROW              0
#define SQL_SCOPE_TRANSACTION         1
#define SQL_SCOPE_SESSION             2
#define SQL_PC_UNKNOWN                 0
#define SQL_PC_NOT_PSEUDO              1
#define SQL_PC_PSEUDO                  2

/* Valid values for connect attribute */
#define SQL_ATTR_AUTO_IPD              10001
#define SQL_ATTR_ACCESS_MODE          10002
#define SQL_ACCESS_MODE                10002
#define SQL_ATTR_AUTOCOMMIT           10003
#define SQL_AUTOCOMMIT                 10003
#define SQL_ATTR_DBC_SYS_NAMING       10004
#define SQL_ATTR_DBC_DEFAULT_LIB      10005
#define SQL_ATTR_ADOPT_OWNER_AUTH     10006

```

```

#define SQL_ATTR_SYSBAS_CMT      10007
#define SQL_ATTR_COMMIT         0
#define SQL_MODE_READ_ONLY     0
#define SQL_MODE_READ_WRITE    1
#define SQL_MODE_DEFAULT       1
#define SQL_AUTOCOMMIT_OFF      0
#define SQL_AUTOCOMMIT_ON      1
#define SQL_TXN_ISOLATION       0
#define SQL_ATTR_TXN_ISOLATION  0
#define SQL_COMMIT_NONE        1
#define SQL_TXN_NO_COMMIT       1
#define SQL_TXN_NOCOMMIT        1
#define SQL_COMMIT_CHG         2
#define SQL_COMMIT_UR           2
#define SQL_TXN_READ_UNCOMMITTED 2
#define SQL_COMMIT_CS           3
#define SQL_TXN_READ_COMMITTED  3
#define SQL_COMMIT_ALL          4
#define SQL_COMMIT_RS           4
#define SQL_TXN_REPEATABLE_READ 4
#define SQL_COMMIT_RR           5
#define SQL_TXN_SERIALIZABLE    5

/* Valid index flags */
#define SQL_INDEX_UNIQUE        0
#define SQL_INDEX_ALL           1
#define SQL_INDEX_OTHER         3
#define SQL_TABLE_STAT          0
#define SQL_ENSURE              1
#define SQL_QUICK                0

/* Valid trace values */
#define SQL_ATTR_TRACE_CLI      1
#define SQL_ATTR_TRACE_DBMON    2
#define SQL_ATTR_TRACE_DEBUG    4
#define SQL_ATTR_TRACE_JOBLOG   8
#define SQL_ATTR_TRACE_STRTRC   16

/* Valid File Options */
#define SQL_FILE_READ           2
#define SQL_FILE_CREATE         8
#define SQL_FILE_OVERWRITE      16
#define SQL_FILE_APPEND         32

/* Valid types for GetDiagField */
#define SQL_DIAG_RETURNCODE     1
#define SQL_DIAG_NUMBER         2
#define SQL_DIAG_ROW_COUNT      3
#define SQL_DIAG_SQLSTATE       4
#define SQL_DIAG_NATIVE         5
#define SQL_DIAG_MESSAGE_TEXT   6
#define SQL_DIAG_DYNAMIC_FUNCTION 7
#define SQL_DIAG_CLASS_ORIGIN   8
#define SQL_DIAG_SUBCLASS_ORIGIN 9
#define SQL_DIAG_CONNECTION_NAME 10
#define SQL_DIAG_SERVER_NAME    11
#define SQL_DIAG_MESSAGE_TOKENS 12
#define SQL_DIAG_AUTOGEN_KEY    14

/*
 * SQLColAttributes defines
 * These are also used by SQLGetInfo
 */
#define SQL_UNSEARCHABLE        0
#define SQL_LIKE_ONLY           1
#define SQL_ALL_EXCEPT_LIKE   2
#define SQL_SEARCHABLE          3

```

```

/* GetFunctions() values to identify CLI functions */
#define SQL_API_SQLALLOCCONNECT 1
#define SQL_API_SQLALLOCENV 2
#define SQL_API_SQLALLOCHANDLE 1001
#define SQL_API_SQLALLOCSTMT 3
#define SQL_API_SQLBINDCOL 4
#define SQL_API_SQLBINDFILETOCOL 2002
#define SQL_API_SQLBINDFILETOPARAM 2003
#define SQL_API_SQLBINDPARAM 1002
#define SQL_API_SQLBINDPARAMETER 1023
#define SQL_API_SQLCANCEL 5
#define SQL_API_SQLCLOSECURSOR 1003
#define SQL_API_SQLCOLATTRIBUTES 6
#define SQL_API_SQLCOLUMNPRIVILEGES 2010
#define SQL_API_SQLCOLUMNS 40
#define SQL_API_SQLCONNECT 7
#define SQL_API_SQLCOPYDESC 1004
#define SQL_API_SQLDATASOURCES 57
#define SQL_API_SQLDESCRIBECOL 8
#define SQL_API_SQLDESCRIBEPARAM 58
#define SQL_API_SQLDISCONNECT 9
#define SQL_API_SQLDRIVERCONNECT 68
#define SQL_API_SQLENDTRAN 1005
#define SQL_API_SQLERROR 10
#define SQL_API_SQLEXECDIRECT 11
#define SQL_API_SQLEXECUTE 12
#define SQL_API_SQLEXTENDEDFETCH 1022
#define SQL_API_SQLFETCH 13
#define SQL_API_SQLFETCHSCROLL 1021
#define SQL_API_SQLFOREIGNKEYS 60
#define SQL_API_SQLFREECONNECT 14
#define SQL_API_SQLFREEENV 15
#define SQL_API_SQLFREEHANDLE 1006
#define SQL_API_SQLFREESTMT 16
#define SQL_API_SQLGETCOL 43
#define SQL_API_SQLGETCONNECTATTR 1007
#define SQL_API_SQLGETCONNECTOPTION 42
#define SQL_API_SQLGETCURSORNAME 17
#define SQL_API_SQLGETDATA 43
#define SQL_API_SQLGETDESCFIELD 1008
#define SQL_API_SQLGETDESCREC 1009
#define SQL_API_SQLGETDIAGFIELD 1010
#define SQL_API_SQLGETDIAGREC 1011
#define SQL_API_SQLGETENVATTR 1012
#define SQL_API_SQLGETFUNCTIONS 44
#define SQL_API_SQLGETINFO 45
#define SQL_API_SQLGETLENGTH 2004
#define SQL_API_SQLGETPOSITION 2005
#define SQL_API_SQLGETSTMTATTR 1014
#define SQL_API_SQLGETSTMTOPTION 46
#define SQL_API_SQLGETSUBSTRING 2006
#define SQL_API_SQLGETTYPEINFO 47
#define SQL_API_SQLLANGUAGES 2001
#define SQL_API_SQLMORERESULTS 61
#define SQL_API_SQLNATIVESQL 62
#define SQL_API_SQLNEXTRESULT 2009
#define SQL_API_SQLNUMPARAMS 63
#define SQL_API_SQLNUMRESULTCOLS 18
#define SQL_API_SQLPARAMDATA 48
#define SQL_API_SQLPARAMOPTIONS 2007
#define SQL_API_SQLPREPARE 19
#define SQL_API_SQLPRIMARYKEYS 65
#define SQL_API_SQLPROCEDURECOLUMNS 66
#define SQL_API_SQLPROCEDURES 67
#define SQL_API_SQLPUTDATA 49
#define SQL_API_SQLRELEASEENV 1015

```



```

#define SQL_API_SQLROWCOUNT      20
#define SQL_API_SQLSETCONNECTATTR 1016
#define SQL_API_SQLSETCONNECTOPTION 50
#define SQL_API_SQLSETCURSORNAME  21
#define SQL_API_SQLSETDESCFIELD    1017
#define SQL_API_SQLSETDESCREC      1018
#define SQL_API_SQLSETENVATTR      1019
#define SQL_API_SQLSETPARAM        22
#define SQL_API_SQLSETSTMTATTR     1020
#define SQL_API_SQLSETSTMTOPTION   51
#define SQL_API_SQLSPECIALCOLUMNS 52
#define SQL_API_SQLSTARTTRAN       2008
#define SQL_API_SQLSTATISTICS      53
#define SQL_API_SQLTABLEPRIVILEGES 2011
#define SQL_API_SQLTABLES          54
#define SQL_API_SQLTRANSACT        23

/* unsupported APIs */
#define SQL_API_SQLSETPOS          -1

/* NULL handle defines */
#ifdef __64BIT__
#define SQL_NULL_HENV              0
#define SQL_NULL_HDBC              0
#define SQL_NULL_HSTMT            0
#else
#define SQL_NULL_HENV              0L
#define SQL_NULL_HDBC              0L
#define SQL_NULL_HSTMT            0L
#endif

#ifdef __64BIT__
#if !defined(SDWORD)
typedef int          SDWORD;
#endif
#if !defined(UDWORD)
typedef unsigned int UDWORD;
#endif
#else
#if !defined(SDWORD)
typedef long int     SDWORD;
#endif
#if !defined(UDWORD)
typedef unsigned long int UDWORD;
#endif
#endif
#if !defined(UWORD)
typedef unsigned short int UWORD;
#endif
#if !defined(SWORD)
typedef signed short int  SWORD;
#endif

typedef char          SQLCHAR;
typedef short int     SQLSMALLINT;
typedef UWORD         SQLUSMALLINT;
typedef UDWORD        SQLINTEGER;
typedef double        SQLDOUBLE;
typedef float         SQLREAL;

typedef void *        PTR;
typedef PTR           SQLPOINTER;

#ifdef __64BIT__
typedef int           SQLINTEGER;
typedef int           HENV;
typedef int           HDBC;

```

```

typedef int          HSTMT;
typedef int          HDESC;
typedef int          SQLHANDLE;
#else
typedef long        int    SQLINTEGER;
typedef long        HENV;
typedef long        HDBC;
typedef long        HSTMT;
typedef long        HDESC;
typedef long        SQLHANDLE;
#endif

typedef HENV        SQLHENV;
typedef HDBC        SQLHDBC;
typedef HSTMT       SQLHSTMT;
typedef HDESC       SQLHDESC;

typedef SQLINTEGER  RETCODE;
typedef RETCODE    SQLRETURN;

typedef float       SFLOAT;

typedef SQLPOINTER  SQLHWND;

/*
 * DATE, TIME, and TIMESTAMP structures. These are for compatibility
 * purposes only. When actually specifying or retrieving DATE, TIME,
 * and TIMESTAMP values, character strings must be used.
 */

typedef struct DATE_STRUCT
{
    SQLSMALLINT    year;
    SQLSMALLINT    month;
    SQLSMALLINT    day;
} DATE_STRUCT;

typedef struct TIME_STRUCT
{
    SQLSMALLINT    hour;
    SQLSMALLINT    minute;
    SQLSMALLINT    second;
} TIME_STRUCT;

typedef struct TIMESTAMP_STRUCT
{
    SQLSMALLINT    year;
    SQLSMALLINT    month;
    SQLSMALLINT    day;
    SQLSMALLINT    hour;
    SQLSMALLINT    minute;
    SQLSMALLINT    second;
    SQLINTEGER     fraction;    /* fraction of a second */
} TIMESTAMP_STRUCT;

/* Transaction info structure */
typedef struct TXN_STRUCT {
    SQLINTEGER     operation;
    SQLCHAR        tminfo[10];
    SQLCHAR        reserved1[2];
    void           *XID;
    SQLINTEGER     timeoutval;
    SQLINTEGER     locktimeout;
}

```

```

    SQLCHAR    reserved2[8];
} TXN_STRUCT;

```

```

SQL_EXTERN SQLRETURN SQLAllocConnect (SQLHENV          henv,
                                       SQLHDBC          *phdbc);

SQL_EXTERN SQLRETURN SQLAllocEnv      (SQLHENV          *phenv);

SQL_EXTERN SQLRETURN SQLAllocHandle  (SQLSMALLINT      htype,
                                       SQLINTEGER        ihnd,
                                       SQLINTEGER        *ohnd);

SQL_EXTERN SQLRETURN SQLAllocStmt    (SQLHDBC          hdbc,
                                       SQLHSTMT          *phstmt);

SQL_EXTERN SQLRETURN SQLBindCol      (SQLHSTMT        hstmt,
                                       SQLSMALLINT      icol,
                                       SQLSMALLINT      iType,
                                       SQLPOINTER       rgbValue,
                                       SQLINTEGER        cbValueMax,
                                       SQLINTEGER        *pcbValue);

SQL_EXTERN SQLRETURN SQLBindFileToCol (SQLHSTMT        hstmt,
                                       SQLSMALLINT      icol,
                                       SQLCHAR          *fName,
                                       SQLSMALLINT      *fNameLen,
                                       SQLINTEGER        *fOptions,
                                       SQLSMALLINT      fValueMax,
                                       SQLINTEGER        *sLen,
                                       SQLINTEGER        *pcbValue);

SQL_EXTERN SQLRETURN SQLBindFileToParam (SQLHSTMT      hstmt,
                                       SQLSMALLINT      ipar,
                                       SQLSMALLINT      iType,
                                       SQLCHAR          *fName,
                                       SQLSMALLINT      *fNameLen,
                                       SQLINTEGER        *fOptions,
                                       SQLSMALLINT      fValueMax,
                                       SQLINTEGER        *pcbValue);

SQL_EXTERN SQLRETURN SQLBindParam    (SQLHSTMT        hstmt,
                                       SQLSMALLINT      iparm,
                                       SQLSMALLINT      iType,
                                       SQLSMALLINT      pType,
                                       SQLINTEGER        pLen,
                                       SQLSMALLINT      pScale,
                                       SQLPOINTER       pData,
                                       SQLINTEGER        *pcbValue);

SQL_EXTERN SQLRETURN SQLBindParameter (SQLHSTMT        hstmt,
                                       SQLSMALLINT      ipar,
                                       SQLSMALLINT      fParamType,
                                       SQLSMALLINT      fCType,
                                       SQLSMALLINT      fSQLType,
                                       SQLINTEGER        pLen,
                                       SQLSMALLINT      pScale,
                                       SQLPOINTER       pData,
                                       SQLINTEGER        cbValueMax,
                                       SQLINTEGER        *pcbValue);

SQL_EXTERN SQLRETURN SQLCancel      (SQLHSTMT        hstmt);

```

```

SQL_EXTERN SQLRETURN SQLCloseCursor (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN SQLColAttributes (SQLHSTMT          hstmt,
                                       SQLSMALLINT       icol,
                                       SQLSMALLINT       fDescType,
                                       SQLCHAR             *rgbDesc,
                                       SQLINTEGER         cbDescMax,
                                       SQLINTEGER         *pcbDesc,
                                       SQLINTEGER         *pfDesc);

SQL_EXTERN SQLRETURN SQLColumnPrivileges (SQLHSTMT       hstmt,
                                           SQLCHAR        *szTableQualifier,
                                           SQLSMALLINT     cbTableQualifier,
                                           SQLCHAR        *szTableOwner,
                                           SQLSMALLINT     cbTableOwner,
                                           SQLCHAR        *szTableName,
                                           SQLSMALLINT     cbTableName,
                                           SQLCHAR        *szColumnName,
                                           SQLSMALLINT     cbColumnName);

SQL_EXTERN SQLRETURN SQLColumns (SQLHSTMT          hstmt,
                                  SQLCHAR           *szTableQualifier,
                                  SQLSMALLINT       cbTableQualifier,
                                  SQLCHAR           *szTableOwner,
                                  SQLSMALLINT       cbTableOwner,
                                  SQLCHAR           *szTableName,
                                  SQLSMALLINT       cbTableName,
                                  SQLCHAR           *szColumnName,
                                  SQLSMALLINT       cbColumnName);

SQL_EXTERN SQLRETURN SQLConnect (SQLHDBC           hdbc,
                                  SQLCHAR           *szDSN,
                                  SQLSMALLINT       cbDSN,
                                  SQLCHAR           *szUID,
                                  SQLSMALLINT       cbUID,
                                  SQLCHAR           *szAuthStr,
                                  SQLSMALLINT       cbAuthStr);

SQL_EXTERN SQLRETURN SQLCopyDesc (SQLHDESC        sDesc,
                                   SQLHDESC        tDesc);

SQL_EXTERN SQLRETURN SQLDataSources (SQLHENV       henv,
                                       SQLSMALLINT   fDirection,
                                       SQLCHAR        *szDSN,
                                       SQLSMALLINT   cbDSNMax,
                                       SQLSMALLINT   *pcbDSN,
                                       SQLCHAR        *szDescription,
                                       SQLSMALLINT   cbDescriptionMax,
                                       SQLSMALLINT   *pcbDescription);

SQL_EXTERN SQLRETURN SQLDescribeCol (SQLHSTMT       hstmt,
                                       SQLSMALLINT   icol,
                                       SQLCHAR        *szColName,
                                       SQLSMALLINT   cbColNameMax,
                                       SQLSMALLINT   *pcbColName,
                                       SQLSMALLINT   *pfSqlType,
                                       SQLINTEGER     *pcbColDef,
                                       SQLSMALLINT   *pibScale,
                                       SQLSMALLINT   *pfNullable);

SQL_EXTERN SQLRETURN SQLDescribeParam (SQLHSTMT      hstmt,
                                       SQLSMALLINT   ipar,
                                       SQLSMALLINT   *pfSqlType,
                                       SQLINTEGER     *pcbColDef,
                                       SQLSMALLINT   *pibScale,
                                       SQLSMALLINT   *pfNullable);

```

```

SQL_EXTERN SQLRETURN SQLDisconnect (SQLHDBC          hdbc);
SQL_EXTERN SQLRETURN SQLDriverConnect (SQLHDBC          hdbc,
SQLPOINTER          hwnd,
SQLCHAR          *szConnStrIn,
SQLSMALLINT          cbConnStrIn,
SQLCHAR          *szConnStrOut,
SQLSMALLINT          cbConnStrOutMax,
SQLSMALLINT          *pcbConnStrOut,
SQLSMALLINT          fDriverCompletion);
SQL_EXTERN SQLRETURN SQLEndTran (SQLSMALLINT          htype,
SQLHENV          henv,
SQLSMALLINT          ctype);
SQL_EXTERN SQLRETURN SQLError (SQLHENV          henv,
SQLHDBC          hdbc,
SQLHSTMT          hstmt,
SQLCHAR          *szSqlState,
SQLINTEGER          *pfNativeError,
SQLCHAR          *szErrorMsg,
SQLSMALLINT          cbErrorMsgMax,
SQLSMALLINT          *pcbErrorMsg);
SQL_EXTERN SQLRETURN SQLExecDirect (SQLHSTMT          hstmt,
SQLCHAR          *szSqlStr,
SQLINTEGER          cbSqlStr);
SQL_EXTERN SQLRETURN SQLExecute (SQLHSTMT          hstmt);
SQL_EXTERN SQLRETURN SQLExtendedFetch (SQLHSTMT          hstmt,
SQLSMALLINT          fOrient,
SQLINTEGER          fOffset,
SQLINTEGER          *pcrow,
SQLSMALLINT          *rgfRowStatus);
SQL_EXTERN SQLRETURN SQLFetch (SQLHSTMT          hstmt);
SQL_EXTERN SQLRETURN SQLFetchScroll (SQLHSTMT          hstmt,
SQLSMALLINT          fOrient,
SQLINTEGER          fOffset);
SQL_EXTERN SQLRETURN SQLForeignKeys (SQLHSTMT          hstmt,
SQLCHAR          *szPkTableQualifier,
SQLSMALLINT          cbPkTableQualifier,
SQLCHAR          *szPkTableOwner,
SQLSMALLINT          cbPkTableOwner,
SQLCHAR          *szPkTableName,
SQLSMALLINT          cbPkTableName,
SQLCHAR          *szFkTableQualifier,
SQLSMALLINT          cbFkTableQualifier,
SQLCHAR          *szFkTableOwner,
SQLSMALLINT          cbFkTableOwner,
SQLCHAR          *szFkTableName,
SQLSMALLINT          cbFkTableName);
SQL_EXTERN SQLRETURN SQLFreeConnect (SQLHDBC          hdbc);
SQL_EXTERN SQLRETURN SQLFreeEnv (SQLHENV          henv);
SQL_EXTERN SQLRETURN SQLFreeStmt (SQLHSTMT          hstmt,
SQLSMALLINT          fOption);
SQL_EXTERN SQLRETURN SQLFreeHandle (SQLSMALLINT          htype,
SQLINTEGER          hndl);
SQL_EXTERN SQLRETURN SQLGetCol (SQLHSTMT          hstmt,

```

```

        SQLSMALLINT      icol,
        SQLSMALLINT      itype,
        SQLPOINTER       tval,
        SQLINTEGER       blen,
        SQLINTEGER       *olen);

SQL_EXTERN SQLRETURN SQLGetConnectAttr (SQLHDBC      hdbc,
        SQLINTEGER      attr,
        SQLPOINTER      oval,
        SQLINTEGER      ilen,
        SQLINTEGER      *olen);

SQL_EXTERN SQLRETURN SQLGetConnectOption (SQLHDBC    hdbc,
        SQLSMALLINT     iopt,
        SQLPOINTER      oval);

SQL_EXTERN SQLRETURN SQLGetCursorName (SQLHSTMT     hstmt,
        SQLCHAR         *szCursor,
        SQLSMALLINT     cbCursorMax,
        SQLSMALLINT     *pcbCursor);

SQL_EXTERN SQLRETURN SQLGetData      (SQLHSTMT     hstmt,
        SQLSMALLINT     icol,
        SQLSMALLINT     fCType,
        SQLPOINTER      rgbValue,
        SQLINTEGER      cbValueMax,
        SQLINTEGER      *pcbValue);

SQL_EXTERN SQLRETURN SQLGetDescField (SQLHDESC     hdesc,
        SQLSMALLINT     rcdNum,
        SQLSMALLINT     fieldID,
        SQLPOINTER      fValue,
        SQLINTEGER      fLength,
        SQLINTEGER      *stLength);

SQL_EXTERN SQLRETURN SQLGetDescRec   (SQLHDESC     hdesc,
        SQLSMALLINT     rcdNum,
        SQLCHAR         *fname,
        SQLSMALLINT     bufLen,
        SQLSMALLINT     *sLength,
        SQLSMALLINT     *sType,
        SQLSMALLINT     *sbType,
        SQLINTEGER      *fLength,
        SQLSMALLINT     *fprec,
        SQLSMALLINT     *fscale,
        SQLSMALLINT     *fnull);

SQL_EXTERN SQLRETURN SQLGetDiagField (SQLSMALLINT  hType,
        SQLINTEGER      hndl,
        SQLSMALLINT     rcdNum,
        SQLSMALLINT     diagID,
        SQLPOINTER      dValue,
        SQLSMALLINT     bLength,
        SQLSMALLINT     *sLength);

SQL_EXTERN SQLRETURN SQLGetDiagRec   (SQLSMALLINT  hType,
        SQLINTEGER      hndl,
        SQLSMALLINT     rcdNum,
        SQLCHAR         *SQLstate,
        SQLINTEGER      *SQLcode,
        SQLCHAR         *msgText,
        SQLSMALLINT     bLength,
        SQLSMALLINT     *SLength);

SQL_EXTERN SQLRETURN SQLGetEnvAttr   (SQLHENV      hEnv,
        SQLINTEGER      fAttribute,
        SQLPOINTER      pParam,

```

```

        SQLINTEGER cbParamMax,
        SQLINTEGER * pcbParam);

SQL_EXTERN SQLRETURN SQLGetFunctions (SQLHDBC          hdbc,
                                       SQLSMALLINT     fFunction,
                                       SQLSMALLINT     *pfExists);

SQL_EXTERN SQLRETURN SQLGetInfo      (SQLHDBC          hdbc,
                                       SQLSMALLINT     fInfoType,
                                       SQLPOINTER      rgbInfoValue,
                                       SQLSMALLINT     cbInfoValueMax,
                                       SQLSMALLINT     *pcbInfoValue);

SQL_EXTERN SQLRETURN SQLGetLength (SQLHSTMT          hstmt,
                                       SQLSMALLINT     locType,
                                       SQLINTEGER      locator,
                                       SQLINTEGER      *sLength,
                                       SQLINTEGER      *ind);

SQL_EXTERN SQLRETURN SQLGetPosition (SQLHSTMT          hstmt,
                                       SQLSMALLINT     locType,
                                       SQLINTEGER      srceLocator,
                                       SQLINTEGER      srchLocator,
                                       SQLCHAR         *srchLiteral,
                                       SQLINTEGER      srchLiteralLen,
                                       SQLINTEGER      fPosition,
                                       SQLINTEGER      *located,
                                       SQLINTEGER      *ind);

SQL_EXTERN SQLRETURN SQLGetStmtAttr (SQLHSTMT          hstmt,
                                       SQLINTEGER      fAttr,
                                       SQLPOINTER      pvParam,
                                       SQLINTEGER      bLength,
                                       SQLINTEGER      *sLength);

SQL_EXTERN SQLRETURN SQLGetStmtOption (SQLHSTMT          hstmt,
                                       SQLSMALLINT     fOption,
                                       SQLPOINTER      pvParam);

SQL_EXTERN SQLRETURN SQLGetSubString (SQLHSTMT          hstmt,
                                       SQLSMALLINT     locType,
                                       SQLINTEGER      srceLocator,
                                       SQLINTEGER      fPosition,
                                       SQLINTEGER      length,
                                       SQLSMALLINT     tType,
                                       SQLPOINTER      rgbValue,
                                       SQLINTEGER      cbValueMax,
                                       SQLINTEGER      *StringLength,
                                       SQLINTEGER      *ind);

SQL_EXTERN SQLRETURN SQLGetTypeInfo (SQLHSTMT          hstmt,
                                       SQLSMALLINT     fSqlType);

SQL_EXTERN SQLRETURN SQLLanguages   (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN SQLMoreResults (SQLHSTMT          hstmt);

SQL_EXTERN SQLRETURN SQLNativeSql  (SQLHDBC          hdbc,
                                       SQLCHAR         *szSqlStrIn,
                                       SQLINTEGER      cbSqlStrIn,
                                       SQLCHAR         *szSqlStr,
                                       SQLINTEGER      cbSqlStrMax,
                                       SQLINTEGER      *pcbSqlStr);

SQL_EXTERN SQLRETURN SQLNextResult (SQLHSTMT          hstmt,
                                       SQLHSTMT         hstmt2);

```

SQL_EXTERN	SQLRETURN	SQLNumParams	(SQLHSTMT SQLSMALLINT	hstmt, *ppcpar);
SQL_EXTERN	SQLRETURN	SQLNumResultCols	(SQLHSTMT SQLSMALLINT	hstmt, *ppccol);
SQL_EXTERN	SQLRETURN	SQLParamData	(SQLHSTMT SQLPOINTER	hstmt, *Value);
SQL_EXTERN	SQLRETURN	SQLParamOptions	(SQLHSTMT SQLINTEGER SQLINTEGER	hstmt, crow, *pirow);
SQL_EXTERN	SQLRETURN	SQLPrepare	(SQLHSTMT SQLCHAR SQLSMALLINT	hstmt, *szSqlStr, cbSqlStr);
SQL_EXTERN	SQLRETURN	SQLPrimaryKeys	(SQLHSTMT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT	hstmt, *szTableQualifier, cbTableQualifier, *szTableOwner, cbTableOwner, *szTableName, cbTableName);
SQL_EXTERN	SQLRETURN	SQLProcedureColumns	(SQLHSTMT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT	hstmt, *szProcQualifier, cbProcQualifier, *szProcOwner, cbProcOwner, *szProcName, cbProcName, *szColumnName, cbColumnName);
SQL_EXTERN	SQLRETURN	SQLProcedures	(SQLHSTMT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT SQLCHAR SQLSMALLINT	hstmt, *szProcQualifier, cbProcQualifier, *szProcOwner, cbProcOwner, *szProcName, cbProcName);
SQL_EXTERN	SQLRETURN	SQLPutData	(SQLHSTMT SQLPOINTER SQLINTEGER	hstmt, Data, SLen);
SQL_EXTERN	SQLRETURN	SQLReleaseEnv	(SQLHENV	henv);
SQL_EXTERN	SQLRETURN	SQLRowCount	(SQLHSTMT SQLINTEGER	hstmt, *pcrow);
SQL_EXTERN	SQLRETURN	SQLSetConnectAttr	(SQLHDBC SQLINTEGER SQLPOINTER SQLINTEGER	hdbc, attrib, vParam, inlen);
SQL_EXTERN	SQLRETURN	SQLSetConnectOption	(SQLHDBC SQLSMALLINT SQLPOINTER	hdbc, fOption, vParam);
SQL_EXTERN	SQLRETURN	SQLSetCursorName	(SQLHSTMT SQLCHAR SQLSMALLINT	hstmt, *szCursor, cbCursor);
SQL_EXTERN	SQLRETURN	SQLSetDescField	(SQLHDESC SQLSMALLINT	hdesc, rcdNum,



```

        SQLSMALLINT    fID,
        SQLPOINTER     Value,
        SQLINTEGER     buffLen);

SQL_EXTERN SQLRETURN SQLSetDescRec (SQLHDESC          hdesc,
        SQLSMALLINT    rcdNum,
        SQLSMALLINT    Type,
        SQLSMALLINT    subType,
        SQLINTEGER     fLength,
        SQLSMALLINT    fPrec,
        SQLSMALLINT    fScale,
        SQLPOINTER     Value,
        SQLINTEGER     *sLength,
        SQLINTEGER     *indic);

SQL_EXTERN SQLRETURN SQLSetEnvAttr( SQLHENV hEnv,
        SQLINTEGER fAttribute,
        SQLPOINTER pParam,
        SQLINTEGER cbParam);

SQL_EXTERN SQLRETURN SQLSetParam (SQLHSTMT          hstmt,
        SQLSMALLINT    ipar,
        SQLSMALLINT    fCType,
        SQLSMALLINT    fSqlType,
        SQLINTEGER     cbColDef,
        SQLSMALLINT    ibScale,
        SQLPOINTER     rgbValue,
        SQLINTEGER     *pcbValue);

SQL_EXTERN SQLRETURN SQLSetStmtAttr (SQLHSTMT          hstmt,
        SQLINTEGER     fAttr,
        SQLPOINTER     pParam,
        SQLINTEGER     vParam);

SQL_EXTERN SQLRETURN SQLSetStmtOption (SQLHSTMT          hstmt,
        SQLSMALLINT    fOption,
        SQLPOINTER     vParam);

SQL_EXTERN SQLRETURN SQLSpecialColumns (SQLHSTMT          hstmt,
        SQLSMALLINT    fColType,
        SQLCHAR         *szTableQual,
        SQLSMALLINT    cbTableQual,
        SQLCHAR         *szTableOwner,
        SQLSMALLINT    cbTableOwner,
        SQLCHAR         *szTableName,
        SQLSMALLINT    cbTableName,
        SQLSMALLINT    fScope,
        SQLSMALLINT    fNullable);

SQL_EXTERN SQLRETURN SQLStartTran (SQLSMALLINT          htype,
        SQLHENV         henv,
        SQLINTEGER     mode,
        SQLINTEGER     clevel);

SQL_EXTERN SQLRETURN SQLStatistics (SQLHSTMT          hstmt,
        SQLCHAR         *szTableQualifier,
        SQLSMALLINT    cbTableQualifier,
        SQLCHAR         *szTableOwner,
        SQLSMALLINT    cbTableOwner,
        SQLCHAR         *szTableName,
        SQLSMALLINT    cbTableName,
        SQLSMALLINT    fUnique,
        SQLSMALLINT    fres);

SQL_EXTERN SQLRETURN SQLTablePrivileges (SQLHSTMT          hstmt,
        SQLCHAR         *szTableQualifier,
        SQLSMALLINT    cbTableQualifier,

```

```

        SQLCHAR      *szTableOwner,
        SQLSMALLINT  cbTableOwner,
        SQLCHAR      *szTableName,
        SQLSMALLINT  cbTableName);

SQL_EXTERN SQLRETURN SQLTables      (SQLHSTMT      hstmt,
        SQLCHAR      *szTableQualifier,
        SQLSMALLINT  cbTableQualifier,
        SQLCHAR      *szTableOwner,
        SQLSMALLINT  cbTableOwner,
        SQLCHAR      *szTableName,
        SQLSMALLINT  cbTableName,
        SQLCHAR      *szTableType,
        SQLSMALLINT  cbTableType);

SQL_EXTERN SQLRETURN SQLTransact    (SQLHENV      henv,
        SQLHDBC      hdbc,
        SQLSMALLINT  fType);

#define FAR
#define SQL_SQLSTATE_SIZE      5 /* size of SQLSTATE, not including
        null terminating byte */
#define SQL_MAX_DSN_LENGTH     18 /* maximum data source name size */
#define SQL_MAX_ID_LENGTH     18 /* maximum identifier name size,
        for example, cursor names */

#define SQL_MAX_STMT_SIZ      65480 /* Maximum statement size */
#define SQL_MAXRECL           32766 /* Maximum record length */

#define SQL_SMALL_LENGTH      2 /* Size of a SMALLINT */
#define SQL_MAXSMALLVAL       32767 /* Maximum value of a SMALLINT */
#define SQL_MINSMALLVAL      ((SQL_MAXSMALLVAL)-1) /* Minimum value of a SMALLINT */
#define SQL_INT_LENGTH        4 /* Size of an INTEGER */
#define SQL_MAXINTVAL         2147483647 /* Maximum value of an INTEGER */
#define SQL_MININTVAL        ((SQL_MAXINTVAL)-1) /* Minimum value of an INTEGER */
#define SQL_FLOAT_LENGTH      8 /* Size of a FLOAT */
#define SQL_DEFDEC_PRECISION  5 /* Default precision for DECIMAL */
#define SQL_DEFDEC_SCALE     0 /* Default scale for DECIMAL */
#define SQL_MAXDECIMAL        31 /* Maximum scale/prec. for DECIMAL */
#define SQL_DEFCHAR           1 /* Default length for a CHAR */
#define SQL_DEFWCHAR          1 /* Default length for a wchar_t */
#define SQL_MAXCHAR           32766 /* Maximum length of a CHAR */
#define SQL_MAXLSTR           255 /* Maximum length of an LSTRING */
#define SQL_MAXVCHAR          32740 /* Maximum length of a
        /* VARCHAR
#define SQL_MAXVGRAPH         16370 /* Maximum length of a VARGRAPHIC
#define SQL_MAXBLOB           2147483647 /* Max. length of a BLOB host var
#define SQL_MAXCLOB           2147483647 /* Max. length of a CLOB host var
#define SQL_MAXDBCLOB         1073741823 /* Max. length of an DBCLOB host
        /* var
#define SQL_LONGMAX           32740 /* Maximum length of a LONG VARCHAR
#define SQL_LONGGRMAX         16370 /* Max. length of a LONG VARGRAPHIC
#define SQL_LVCHAROH          26 /* Overhead for LONG VARCHAR in
        /* record
#define SQL_LOBCHAROH         312 /* Overhead for LOB in record
#define SQL_BLOB_MAXLEN       2147483647 /* BLOB maximum length, in bytes
#define SQL_CLOB_MAXLEN       2147483647 /* CLOB maximum length, in chars
#define SQL_DBCLOB_MAXLEN     1073741823 /* maxlen for dbclob
#define SQL_TIME_LENGTH       3 /* Size of a TIME field
#define SQL_TIME_STRLEN       8 /* Size of a TIME field output
#define SQL_TIME_MINSTRLEN    5 /* Size of a non-USA TIME field
        /* output without seconds
#define SQL_DATE_LENGTH       4 /* Size of a DATE field
#define SQL_DATE_STRLEN       10 /* Size of a DATE field output
#define SQL_STAMP_LENGTH      10 /* Size of a TIMESTAMP field

```

```

#define SQL_STAMP_STRLEN      26      /* Size of a TIMESTAMP field output */
#define SQL_STAMP_MINSTRLEN  19      /* Size of a TIMESTAMP field output
                                     /* without microseconds */
#define SQL_BOOLEAN_LENGTH  1        /* Size of a BOOLEAN field */
#define SQL_IND_LENGTH      2        /* Size of an indicator value */

#define SQL_MAX_PNAME_LENGTH 254     /* Max size of Stored Proc Name */
#define SQL_LG_IDENT        18      /* Maximum length of Long Identifier */
#define SQL_SH_IDENT        8        /* Maximum length of Short Identifier */
#define SQL_MN_IDENT        1        /* Minimum length of Identifiers */
#define SQL_MAX_VAR_NAME    30      /* Max size of Host Variable Name */
#define SQL_KILO_VALUE      1024    /* # of bytes in a kilobyte */
#define SQL_MEGA_VALUE      1048576 /* # of bytes in a megabyte */
#define SQL_GIGA_VALUE      1073741824 /* # of bytes in a gigabyte */

/* SQL extended data types (negative means unsupported) */
#define SQL_TINYINT          -6
#define SQL_BIT              -7

/* C data type to SQL data type mapping */
#define SQL_C_CHAR           SQL_CHAR /* CHAR, VARCHAR, DECIMAL, NUMERIC */
#define SQL_C_LONG           SQL_INTEGER /* INTEGER */
#define SQL_C_SLONG          SQL_INTEGER /* INTEGER */
#define SQL_C_SHORT          SQL_SMALLINT /* SMALLINT */
#define SQL_C_FLOAT          SQL_REAL /* REAL */
#define SQL_C_DOUBLE         SQL_DOUBLE /* FLOAT, DOUBLE */
#define SQL_C_DATE           SQL_DATE /* DATE */
#define SQL_C_TIME           SQL_TIME /* TIME */
#define SQL_C_TIMESTAMP     SQL_TIMESTAMP /* TIMESTAMP */
#define SQL_C_BINARY         SQL_BINARY /* BINARY, VARBINARY */
#define SQL_C_BIT            SQL_BIT
#define SQL_C_TINYINT        SQL_TINYINT
#define SQL_C_BIGINT         SQL_BIGINT
#define SQL_C_DBCHAR         SQL_DBCLCLOB
#define SQL_C_WCHAR          SQL_WCHAR /* UNICODE */
#define SQL_C_DATETIME       SQL_DATETIME /* DATETIME */
#define SQL_C_BLOB           SQL_BLOB
#define SQL_C_CLOB           SQL_CLOB
#define SQL_C_DBCLCLOB       SQL_DBCLCLOB
#define SQL_C_BLOB_LOCATOR   SQL_BLOB_LOCATOR
#define SQL_C_CLOB_LOCATOR   SQL_CLOB_LOCATOR
#define SQL_C_DBCLCLOB_LOCATOR SQL_DBCLCLOB_LOCATOR

/* miscellaneous constants and unsupported functions */
#define SQL_ADD              -1
#define SQL_ATTR_PARAMSET_SIZE -1
#define SQL_ATTR_PARAMS_PROCESSED_PTR -1
#define SQL_ATTR_PARAM_BIND_TYPE -1
#define SQL_ATTR_PARAM_STATUS_PTR -1
#define SQL_DELETE          -1
#define SQL_KEYSET_SIZE     -1
#define SQL_LCK_NO_CHANGE   -1
#define SQL_LOCK_NO_CHANGE -1
#define SQL_LOCK_EXCLUSIVE  -1
#define SQL_LOCK_UNLOCK     -1
#define SQL_METH_D          -1
#define SQL_POSITION        -1
#define SQL_QUERY_TIMEOUT   -1
#define SQL_ROW_ADDED       -1
#define SQL_ROW_NOROW       -1
#define SQL_ROW_ERROR       -1
#define SQL_ROW_SUCCESS     0
#define SQL_ROW_SUCCESS_WITH_INFO -1
#define SQL_SC_TRY_UNIQUE   -1
#define SQL_SIMULATE_CURSOR -1
#define SQL_UNKNOWN_TYPE    -1
#define SQL_UPDATE          -1

```

```

#define SQL_WARN_VAL_TRUNC                "01004"

#if (__OS400_TGTVRM__ >= 510) /* @B1A*/
#pragma datamodel(pop)        /* @B1A*/
#endif                            /* @B1A*/

#ifndef __ILEC400__
#pragma info(restore)
#endif

#endif /* SQL_H_SQLCLI */

```

---

## Run DB2 UDB CLI in server mode

This topic is about why and how you should run your DB2 UDB CLI applications in server mode.

The reason for running in SQL server mode is that many applications have the need to act as database servers. This means that a single job performs SQL requests on behalf of multiple users. Without using SQL server mode, applications might encounter one or more of the following three limitations:

1. A single job can only have one commit transaction per activation group.
2. A single job can only connect to an RDB once.
3. All SQL statements run under the job's user profile, regardless of the user ID passed in on the connect.

SQL server mode circumvents these limitations by routing all SQL statements to separate jobs. Each connection runs in its own job. The system uses prestart jobs in the QSYSWRK subsystem to minimize the startup time for each connection. Because each call to SQLConnect can accept a different user profile, each job also has its own commit transaction. As soon as the SQLDisconnect has been performed, the job is reset and put back in the pool of available jobs.

## Start DB2 UDB CLI in SQL server mode

There are two ways to place a job into SQL server mode:

1. The most likely case is using the CLI function, *SQLSetEnvAttr*. The SQL server mode is best suited to CLI applications because they already use the concept of multiple connections handles. Set this mode immediately after allocating the CLI environment. Furthermore, the job must not have run any SQL, or started commitment control, before setting this mode. If either one of those cases is true, the mode does not become changed to server mode, and SQL continues to run "inline".

EXAMPLE.

```

.
SQLAllocEnv(&henv);
long attr;
attr = SQL_TRUE
SQLSetEnvAttr(henv,SQL_ATTR_SERVER_MODE,&attr,0);
SQLAllocConnect(henv,&hdbc);
.
.

```

2. The second way to set server mode is using the Change Job (QWTCHGJB) API. Refer to the APIs topic in the information center for a complete description of the QWTCHGJB API.

As soon as SQL server mode has been set, all SQL connections and SQL statements run in server mode. There is no switching back and forth. The job, once in server mode, cannot start commitment control, and cannot use Interactive SQL.

## Restrictions for running DB2 UDB CLI in server mode

- A job must set server mode at the very beginning of processing, before doing anything else. For jobs that are strictly CLI users, they must use the `SQLSetEnvAttr` call to turn on server mode. Remember to do this right after `SQLAL1ocEnv` but before any other calls. As soon as the server mode is on, it cannot be turned off.
- All the SQL functions run in the prestart jobs and commitment control. DO NOT start commitment control in the originating job, either before or after entering server mode.
- Because the SQL is processed in the prestart job, there is no sensitivity to certain changes in the originating job. This includes changes to library list, job priority, message logging, and so forth. The prestart is sensitive to a change of the coded character set identifier (CCSID) value in the originating job, because this can affect the way data is mapped back to the program of the user.
- When running server mode, the application MUST use SQL commits and rollbacks, either embedded or by the SQL CLI. They cannot use the CL commands, because there is no commitment control that is running in the originating job. The job MUST issue a COMMIT before disconnecting; otherwise an implicit ROLLBACK occurs.
- It is not possible to use interactive SQL from a job in server mode. Use of STRSQL when in server mode results in an SQL6141 message.
- It is also not possible to perform SQL compiles while in server mode. Server mode can be used when running compiled SQL programs, but must not be on for the compiles. The compiles fail, if the job is in server mode.
- `SQLDataSources` is unique in that it does not require a connection handle to run. When in server mode, the program must already have done a connect to the local database, before using `SQLDataSources`. Since `DataSources` is used to find the name of the RDB for connection, IBM supports passing a NULL pointer for the RDB name on `SQLConnect`. This obtains a local connection. This makes it possible to write a generic program, when there is no prior knowledge of the system names.
- When doing commits and rollbacks through the CLI, the calls to `SQLEndTran` and `SQLTransact` must include a connection handle. When not running in server mode, one can omit the connection handle to commit everything. However, this is not supported in server mode, because each connection (or thread) has its own transaction scoping.
- It is not recommended to share connection handles across threads, when running in SQL server mode. This is because one thread can overwrite return data or error information that another thread has yet to process.
- If any other SQL work has been done in the job before setting server mode in CLI, then it is impossible to change CLI's environment to run in server mode. An example of this is the use of embedded SQL before the call to do any CLI work that attempted to set the server mode attribute.

---

## Examples: DB2 UDB CLI applications

This topic provides complete examples of DB2 UDB CLI applications.

The examples used through this topic have been drawn from the applications provided in the SQL call level interface topic collection. Detailed error checking has not been implemented in the examples.

### Example: Embedded SQL and the equivalent DB2 UDB CLI function calls

This example shows embedded statements in comments, and the equivalent DB2 UDB CLI function calls.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 260.

```
/******  
** file = embedded.c  
**
```

```

** Example of executing an SQL statement using CLI.
** The equivalent embedded SQL statements are shown in comments.
**
** Functions used:
**
**      SQLAllocConnect      SQLFreeConnect
**      SQLAllocEnv          SQLFreeEnv
**      SQLAllocStmt         SQLFreeStmt
**      SQLConnect           SQLDisconnect
**
**      SQLBindCol           SQLFetch
**      SQLSetParam          SQLTransact
**      SQLError             SQLExecDirect
**
*****/
#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#ifdef NULL
#define NULL 0
#endif

int print_err (SQLHDBC  hdbc,
              SQLHSTMT hstmt);

int main ()
{
    SQLHENV      henv;
    SQLHDBC      hdbc;
    SQLHSTMT     hstmt;

    SQLCHAR      server[] = "sample";
    SQLCHAR      uid[30];
    SQLCHAR      pwd[30];

    SQLINTEGER   id;
    SQLCHAR      name[51];
    SQLINTEGER   namelen, intlen;
    SQLSMALLINT  scale;

    scale = 0;

    /* EXEC SQL CONNECT TO :server USER :uid USING :authentication_string; */
    SQLAllocEnv (&henv);          /* allocate an environment handle */

    SQLAllocConnect (henv, &hdbc); /* allocate a connection handle */

    /* Connect to database indicated by "server" variable with          */
    /* authorization-name given in "uid", authentication-string given   */
    /* in "pwd". Note server, uid, and pwd contain null-terminated     */
    /* strings, as indicated by the 3 input lengths set to SQL_NTS      */
    if (SQLConnect (hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS)
        != SQL_SUCCESS)
        return (print_err (hdbc, SQL_NULL_HSTMT));

    SQLAllocStmt (hdbc, &hstmt); /* allocate a statement handle */

    /* EXEC SQL CREATE TABLE NAMEID (ID integer, NAME varchar(50));   */
    {
        SQLCHAR create[] = "CREATE TABLE NAMEID (ID integer, NAME varchar(50))";

        /* execute the sql statement                                     */
        if (SQLExecDirect (hstmt, create, SQL_NTS) != SQL_SUCCESS)

```

```

        return (print_err (hdbc, hstmt));
    }

/* EXEC SQL COMMIT WORK;                                     */
SQLTransact (henv, hdbc, SQL_COMMIT);                       /* commit create table */

/* EXEC SQL INSERT INTO NAMEID VALUES ( :id, :name         */
{
    SQLCHAR insert[] = "INSERT INTO NAMEID VALUES (?, ?)";

/* show the use of SQLPrepare/SQLExecute method           */
/* prepare the insert                                     */

    if (SQLPrepare (hstmt, insert, SQL_NTS) != SQL_SUCCESS)
        return (print_err (hdbc, hstmt));

/* Set up the first input parameter "id"                 */
    intlen = sizeof (SQLINTEGER);
    SQLSetParam (hstmt, 1,
                SQL_C_LONG, SQL_INTEGER,
                (SQLINTEGER) sizeof (SQLINTEGER),
                scale, (SQLPOINTER) &id,
                (SQLINTEGER *) &intlen);

    namelen = SQL_NTS;
/* Set up the second input parameter "name"              */
    SQLSetParam (hstmt, 2,
                SQL_C_CHAR, SQL_VARCHAR,
                50,
                scale, (SQLPOINTER) name,
                (SQLINTEGER *) &namelen);

/* now assign parameter values and execute the insert     */
    id=500;
    strcpy (name, "Babbage");

    if (SQLExecute (hstmt) != SQL_SUCCESS)
        return (print_err (hdbc, hstmt));
}

/* EXEC SQL COMMIT WORK;                                     */
SQLTransact (henv, hdbc, SQL_COMMIT);                       /* commit inserts */

/* EXEC SQL DECLARE c1 CURSOR FOR SELECT ID, NAME FROM NAMEID; */
/* EXEC SQL OPEN c1;                                       */
/* The application doesn't specify "declare c1 cursor for" */
{
    SQLCHAR select[] = "select ID, NAME from NAMEID";
    if (SQLExecDirect (hstmt, select, SQL_NTS) != SQL_SUCCESS)
        return (print_err (hdbc, hstmt));
}

/* EXEC SQL FETCH c1 INTO :id, :name;                       */
/* Binding first column to output variable "id"           */
SQLBindCol (hstmt, 1,
            SQL_C_LONG, (SQLPOINTER) &id,
            (SQLINTEGER) sizeof (SQLINTEGER),
            (SQLINTEGER *) &intlen);

/* Binding second column to output variable "name"        */
SQLBindCol (hstmt, 2,

```

```

        SQL_C_CHAR, (SQLPOINTER) name,
        (SQLINTEGER) sizeof (name),
        &namelen);

SQLFetch (hstmt);                /* now execute the fetch */
printf("Result of Select: id = %ld name = %s\n", id, name);

/* finally, we should commit, discard hstmt, disconnect */
/* EXEC SQL COMMIT WORK; */
SQLTransact (henv, hdbc, SQL_COMMIT); /* commit the transaction */

/* EXEC SQL CLOSE c1; */
SQLFreeStmt (hstmt, SQL_DROP);    /* free the statement handle */

/* EXEC SQL DISCONNECT; */
SQLDisconnect (hdbc);            /* disconnect from the database */

SQLFreeConnect (hdbc);          /* free the connection handle */
SQLFreeEnv (henv);              /* free the environment handle */

return (0);
}

int print_err (SQLHDBC   hdbc,
              SQLHSTMT  hstmt)
{
SQLCHAR      buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR      sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER   sqlcode;
SQLSMALLINT  length;

    while ( SQLError(SQL_NULL_HENV, hdbc, hstmt,
                    sqlstate,
                    &sqlcode,
                    buffer,
                    SQL_MAX_MESSAGE_LENGTH + 1,
                    &length) == SQL_SUCCESS )
    {
        printf("SQLSTATE: %s Native Error Code: %ld\n",
              sqlstate, sqlcode);
        printf("%s \n", buffer);
        printf("----- \n");
    };

    return(SQL_ERROR);
}

```

## Example: Use the CLI XA transaction connection attributes

This example shows how to use the CLI XA transaction connection attributes.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/*****
** file = CLIXAEXMP1.c
**
** Example of a typical flow of work in an XA transaction using the CLI.
**
** XA Functions used:
**
**     xa_open()    -- Open an XA resource for use in a transaction
**     xa_prepare() -- Prepare for commitment of work in the transaction
**     xa_commit()  -- Commit work done in the transaction

```



```

**
** CLI Functions used:
**
**      SQLAllocHandle   SQLBindParameter   SQLDisconnect
**      SQLError         SQLExecute         SQLFreeHandle
**      SQLPrepare       SQLSetConnectAttr  SQLSetEnvAttr
**
** This example will:
** - Open the XA transaction manager
** - Open a CLI connection and start a transaction for it using SQL_TXN_CREATE
** - Do some commitable CLI work under this transaction
** - End the transaction on the first connection using SQL_TXN_END
** - Close the first CLI connection and open a second connection
** - Use the SQL_TXN_FIND option to find the previous transaction
** - Do more commitable work on this transaction and end the transaction
** - Use the XA APIs to prepare and commit the work
**
**/
#define _XA_PROTOTYPES
#define _MULTI_THREADED
#include <xa.h>
#include <stdio.h>
#include <string.h>
#include <sqlcli.h>
#include <time.h>
#include <stdlib.h>

void genXid(XID *xid) {
    time_t    t;
    memset(xid, 0, sizeof(xid));
    xid->formatID = 69;
    xid->gtrid_length = 4;
    xid->bqual_length = 4;
    /* xid->data must be a globally unique naming identifier
       when taking gtrid and bqual together - the example below
       is most likely not unique */
    /* gtrid contents */
    xid->data[0] = 0xFA;
    xid->data[1] = 0xED;
    xid->data[2] = 0xFA;
    xid->data[3] = 0xED;
    time(&t);
    /* bqual contents */
    xid->data[4] = (((int)t) >> 24) & 0xFF;
    xid->data[5] = (((int)t) >> 16) & 0xFF;
    xid->data[6] = (((int)t) >> 8) & 0xFF;
    xid->data[7] = (((int)t) >> 0) & 0xFF;
}

int main(int argc, char **argv)
{
**/
/* Declarations Section */
**/
    SQLHENV  henv;
    SQLHDBC  hdbc;
    SQLHSTMT hstmt;
    SQLRETURN rtnc;
    SQLINTEGER attr;
    SQLINTEGER int_buffer;
    SQLINTEGER rlength;
    SQLINTEGER buffint;
    SQLINTEGER ilen;
    SQLCHAR s[80];
    SQLCHAR state[10];
    SQLCHAR buffer[600];
    SQLCHAR sqlstr[600];
    SQLINTEGER natErr;

```

```

SQLSMALLINT len;

/* Declare local XA variables */
struct TXN_STRUCT new;
XID      xid;
char     xaOpenFormat[128];
int      mainRmid = 1;
int      xaRc;

/* Initialize the XA structure variable's (defined in sqlcli.h) */
strcpy(new.tminfo,"MYPRODUCT");
strcpy(new.reserved1,"");
new.timeoutval = 0;
new.locktimeout = 0;
strcpy(new.reserved2,"");
genXid(&xid);
new.XID = &xid;

/* Use the XA APIs to start the transaction manager */
/* The xa_info argument for xa_open MUST include the THDCTL=C keyword
   and value when using using CLI with XA transactions */
sprintf(xaOpenFormat, "RDBNAME=*LOCAL THDCTL=C");
xaRc = xa_open(xaOpenFormat, mainRmid, TMNOFLAGS);
printf("xa_open(%s, %d, TMNOFLAGS) = %d\n",
       xaOpenFormat, mainRmid, xaRc);

/* Setup the CLI resources */
attr=SQL_TRUE;
rtnc=SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
rtnc=SQLSetEnvAttr(henv,SQL_ATTR_SERVER_MODE,&attr,0); /* set server mode */
rtnc=SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);

/* Mark the connection as an external transaction and connect */
rtnc=SQLSetConnectAttr(hdbc,SQL_ATTR_TXN_EXTERNAL,&attr,0);
rtnc=SQLConnect(hdbc,NULL,0,NULL,0,NULL,0);

/* Start the transaction */
new.operation = SQL_TXN_CREATE;
rtnc=SQLSetConnectAttr(hdbc,SQL_ATTR_TXN_INFO,&new,0);

/* Do some CLI work */
rtnc=SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
strcpy(sqlstr,"insert into tab values(?)");
rtnc=SQLPrepare(hstmt,sqlstr,SQL_NTS);
rtnc=
SQLBindParameter(hstmt,1,1,SQL_INTEGER,SQL_INTEGER,10,2,&buffint,0,&iLen);
buffint=10; /* set the integer value to insert */
rtnc=SQLExecute(hstmt);
if (rtnc!=SQL_SUCCESS)
{
printf("SQLExecute failed with return code: %i \n", rtnc);
rtnc=SQLError(0, 0,hstmt, state, &natErr, buffer, 600, &len);
printf("%i is the SQLCODE\n",natErr);
printf("%i is the length of error text\n",len);
printf("%s is the state\n",state );
printf("%s \n",buffer);
}
else
printf("SQLExecute succeeded, value %i inserted \n", buffint);

/* End the transaction */
new.operation = SQL_TXN_END;
rtnc=SQLSetConnectAttr(hdbc,SQL_ATTR_TXN_INFO,&new,0);

/* Cleanup and disconnect from the first connection */
rtnc=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);
rtnc=SQLDisconnect(hdbc);

```

```

/* Mark the second connection as an external transaction and connect */
attr=SQL_TRUE;
rtnc=SQLSetConnectAttr(hdbc,SQL_ATTR_TXN_EXTERNAL,&attr,0);
rtnc=SQLConnect(hdbc,NULL,0,NULL,0,NULL,0);

/* Find the open transaction from the first connection */
new.operation = SQL_TXN_FIND;
rtnc=SQLSetConnectAttr(hdbc,SQL_ATTR_TXN_INFO,&new,0);

/* Do some CLI work on the second connection */
rtnc=SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hstmt);
strcpy(sqlstr,"insert into tab values(?)");
rtnc=SQLPrepare(hstmt,sqlstr,SQL_NTS);
rtnc=
SQLBindParameter(hstmt,1,1,SQL_INTEGER,SQL_INTEGER,10,2,&buffint,0,&ilen);
buffint=15; /* set the integer value to insert */
rtnc=SQLExecute(hstmt);
if (rtnc!=SQL_SUCCESS)
{
printf("SQLExecute failed with return code: %i \n", rtnc);
rtnc=SQLError(0, 0,hstmt, state, &natErr, buffer, 600, &len);
printf("%i is the SQLCODE\n",natErr);
printf("%i is the length of error text\n",len);
printf("%s is the state\n",state );
printf("%s \n",buffer);
}
else
printf("Second SQLExecute succeeded, value %i inserted \n", buffint);

/* End the transaction */
new.operation = SQL_TXN_END;
rtnc=SQLSetConnectAttr(hdbc,SQL_ATTR_TXN_INFO,&new,0);

/* Now, use XA to prepare/commit transaction */
/* Prepare to commit */
xaRc = xa_prepare(&xid, mainRmid, TMNOFLAGS);
printf("xa_prepare(xid, %d, TMNOFLAGS) = %d\n",mainRmid, xaRc);

/* Commit */
if (xaRc != XA_RDONLY) {
xaRc = xa_commit(&xid, mainRmid, TMNOFLAGS);
printf("xa_commit(xid, %d, TMNOFLAGS) = %d\n", mainRmid, xaRc);
}
else {
printf("xa_commit() skipped for read only TX\n");
}

/* Cleanup the CLI resources */
rtnc=SQLFreeHandle(SQL_HANDLE_STMT,hstmt);
rtnc=SQLDisconnect(hdbc);
rtnc=SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
rtnc=SQLFreeHandle(SQL_HANDLE_ENV,henv);
return 0;
}

```

## Example: Interactive SQL and the equivalent DB2 UDB CLI function calls

This example shows the processing of interactive SQL statements, and follows the flow described in “Write a DB2 UDB CLI application” on page 5.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 260.

```

/*****
** file = typical.c
**
** Example of executing interactive SQL statements, displaying result sets
** and simple transaction management.
**
** Functions used:
**
**      SQLAllocConnect      SQLFreeConnect
**      SQLAllocEnv         SQLFreeEnv
**      SQLAllocStmnt       SQLFreeStmnt
**      SQLConnect          SQLDisconnect
**
**      SQLBindCol          SQLFetch
**      SQLDescribeCol      SQLNumResultCols
**      SQLError            SQLRowCount
**      SQLExecDirect       SQLTransact
**
*****/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "sqlcli.h"

#define MAX_STMT_LEN 255
#define MAXCOLS 100

#define max(a,b) (a > b ? a : b)

int initialize(SQLHENV *henv,
              SQLHDBC *hdbc);

int process_stmt(SQLHENV henv,
                SQLHDBC hdbc,
                SQLCHAR *sqlstr);

int terminate(SQLHENV henv,
              SQLHDBC hdbc);

int print_error(SQLHENV henv,
                SQLHDBC hdbc,
                SQLHSTMT hstmt);

int check_error(SQLHENV henv,
                SQLHDBC hdbc,
                SQLHSTMT hstmt,
                SQLRETURN frc);

void display_results(SQLHSTMT hstmt,
                    SQLSMALLINT nresultcols);

/*****
** main
** - initialize
** - start a transaction
** - get statement
** - another statement?
** - COMMIT or ROLLBACK
** - another transaction?
** - terminate
*****/
int main()
{
    SQLHENV henv;
    SQLHDBC hdbc;
    SQLCHAR sqlstmt[MAX_STMT_LEN + 1]="";

```

```

SQLCHAR    sqltrans[sizeof("ROLLBACK")];
SQLRETURN  rc;

rc = initialize(&henv, &hdbc);
if (rc == SQL_ERROR) return(terminate(henv, hdbc));

printf("Enter an SQL statement to start a transaction(or 'q' to Quit):\n");
gets(sqlstmt);

while (sqlstmt[0] != 'q')
{
    while (sqlstmt[0] != 'q')
    {
        rc = process_stmt(henv, hdbc, sqlstmt);
        if (rc == SQL_ERROR) return(SQL_ERROR);
        printf("Enter an SQL statement(or 'q' to Quit):\n");
        gets(sqlstmt);
    }

    printf("Enter 'c' to COMMIT or 'r' to ROLLBACK the transaction\n");
    fgets(sqltrans, sizeof("ROLLBACK"), stdin);

    if (sqltrans[0] == 'c')
    {
        rc = SQLTransact (henv, hdbc, SQL_COMMIT);
        if (rc == SQL_SUCCESS)
            printf ("Transaction commit was successful\n");
        else
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
    }

    if (sqltrans[0] == 'r')
    {
        rc = SQLTransact (henv, hdbc, SQL_ROLLBACK);
        if (rc == SQL_SUCCESS)
            printf ("Transaction roll back was successful\n");
        else
            check_error (henv, hdbc, SQL_NULL_HSTMT, rc);
    }

    printf("Enter an SQL statement to start a transaction or 'q' to quit\n");
    gets(sqlstmt);
}

terminate(henv, hdbc);

return (SQL_SUCCESS);
}/* end main */

/*****
** process_stmt
** - allocates a statement handle
** - executes the statement
** - determines the type of statement
** - if there are no result columns, therefore non-select statement
**   - if rowcount > 0, assume statement was UPDATE, INSERT, DELETE
**   else
**     - assume a DDL, or Grant/Revoke statement
**   else
**     - must be a select statement.
**     - display results
** - frees the statement handle
*****/

int process_stmt (SQLHENV    henv,
                 SQLHDBC    hdbc,
                 SQLCHAR    *sqlstr)
{

```

```

SQLHSTMT      hstmt;
SQLSMALLINT   nresultcols;
SQLINTEGER    rowcount;
SQLRETURN     rc;

    SQLAllocStmt (hdbc, &hstmt);      /* allocate a statement handle */

    /* execute the SQL statement in "sqlstr" */

rc = SQLExecDirect (hstmt, sqlstr, SQL_NTS);
if (rc != SQL_SUCCESS)
    if (rc == SQL_NO_DATA_FOUND) {
        printf("\nStatement executed without error, however,\n");
        printf("no data was found or modified\n");
        return (SQL_SUCCESS);
    }
    else
        check_error (henv, hdbc, hstmt, rc);

SQLRowCount (hstmt, &rowcount);
rc = SQLNumResultCols (hstmt, &nresultcols);
if (rc != SQL_SUCCESS)
    check_error (henv, hdbc, hstmt, rc);

/* determine statement type */
if (nresultcols == 0) /* statement is not a select statement */
{
    if (rowcount > 0) /* assume statement is UPDATE, INSERT, DELETE */
    {
        printf ("Statement executed, %ld rows affected\n", rowcount);
    }
    else /* assume statement is GRANT, REVOKE or a DLL statement */
    {
        printf ("Statement completed successful\n");
    }
}
else /* display the result set */
{
    display_results(hstmt, nresultcols);
} /* end determine statement type */

SQLFreeStmt (hstmt, SQL_DROP );      /* free statement handle */

return (0);
}/* end process_stmt */

/*****
** initialize
** - allocate environment handle
** - allocate connection handle
** - prompt for server, user id, & password
** - connect to server
*****/

int initialize(SQLHENV *henv,
              SQLHDBC *hdbc)
{
SQLCHAR      server[18],
             uid[10],
             pwd[10];
SQLRETURN    rc;

    rc = SQLAllocEnv (henv);          /* allocate an environment handle */
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

```

```

rc = SQLAllocConnect (*henv, hdbc); /* allocate a connection handle */
if (rc != SQL_SUCCESS )
    check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);

printf("Enter Server Name:\n");
gets(server);
printf("Enter User Name:\n");
gets(uid);
printf("Enter Password Name:\n");
gets(pwd);

if (uid[0] == '\0')
{
    rc = SQLConnect (*hdbc, server, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
}
else
{
    rc = SQLConnect (*hdbc, server, SQL_NTS, uid, SQL_NTS, pwd, SQL_NTS);
    if (rc != SQL_SUCCESS )
        check_error (*henv, *hdbc, SQL_NULL_HSTMT, rc);
}
}/* end initialize */

/*****
** terminate
** - disconnect
** - free connection handle
** - free environment handle
*****/
int terminate(SQLHENV henv,
              SQLHDBC hdbc)
{
SQLRETURN rc;

    rc = SQLDisconnect (hdbc);          /* disconnect from database */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeConnect (hdbc);         /* free connection handle */
    if (rc != SQL_SUCCESS )
        print_error (henv, hdbc, SQL_NULL_HSTMT);
    rc = SQLFreeEnv (henv);            /* free environment handle */
    if (rc != SQL_SUCCESS )
        print_error (henv, SQL_NULL_HDBC, SQL_NULL_HSTMT);

}/* end terminate */

/*****
** display_results - displays the selected character fields
**
** - for each column
**   - get column name
**   - bind column
** - display column headings
** - fetch each row
**   - if value truncated, build error message
**   - if column null, set value to "NULL"
**   - display row
**   - print truncation message
** - free local storage
**
*****/
void display_results(SQLHSTMT hstmt,
                    SQLSMALLINT nresultcols)
{
SQLCHAR colname[32];
SQLSMALLINT coltype[MAXCOLS];
SQLSMALLINT colnamelen;

```

```

SQLSMALLINT    nullable;
SQLINTEGER     collen[MAXCOLS];
SQLSMALLINT    scale;
SQLINTEGER     outlen[MAXCOLS];
SQLCHAR *     data[MAXCOLS];
SQLCHAR        errmsg[256];
SQLRETURN      rc;
SQLINTEGER     i;
SQLINTEGER     displaysize;

for (i = 0; i < nresultcols; i++)
{
    SQLDescribeCol (hstmt, i+1, colname, sizeof (colname),
&colnamelen, &coltype[i], &collen[i], &scale, &nullable);

    /* get display length for column */
    SQLColAttributes (hstmt, i+1, SQL_DESC_PRECISION, NULL, 0
        ,
        NULL, &displaysize);

    /* set column length to max of display length, and column name
        length. Plus one byte for null terminator */
    collen[i] = max(displaysize, collen[i]);
    collen[i] = max(collen[i], strlen((char *) colname) ) + 1;

    printf ("%-*.*s", collen[i], collen[i], colname);

    /* allocate memory to bind column */
    data[i] = (SQLCHAR *) malloc (collen[i]);

    /* bind columns to program vars, converting all types to CHAR */
    SQLBindCol (hstmt, i+1, SQL_C_CHAR, data[i], collen[i], &outlen[i]);
}
printf("\n");

/* display result rows */
while ((rc = SQLFetch (hstmt)) != SQL_NO_DATA_FOUND)
{
    errmsg[0] = '\0';
    for (i = 0; i < nresultcols; i++)
    {
        /* Build a truncation message for any columns truncated */
        if (outlen[i] >= collen[i])
        {
            sprintf ((char *) errmsg + strlen ((char *) errmsg),
                "%d chars truncated, col %d\n",
                outlen[i]-collen[i]+1, i+1);
        }
        if (outlen[i] == SQL_NULL_DATA)
            printf ("%-*.*s", collen[i], collen[i], "NULL");
        else
            printf ("%-*.*s", collen[i], collen[i], data[i]);
    } /* for all columns in this row */

    printf ("\n%s", errmsg); /* print any truncation messages */
} /* while rows to fetch */

/* free data buffers */
for (i = 0; i < nresultcols; i++)
{
    free (data[i]);
}

} /* end display_results

/*****
** SUPPORT FUNCTIONS
** - print_error - call SQLError(), display SQLSTATE and message
** - check_error - call print_error

```



```

**          - check severity of Return Code
**          - rollback & exit if error, continue if warning
*****/

/*****/
int print_error (SQLHENV   henv,
                SQLHDBC   hdbc,
                SQLHSTMT  hstmt)
{
SQLCHAR   buffer[SQL_MAX_MESSAGE_LENGTH + 1];
SQLCHAR   sqlstate[SQL_SQLSTATE_SIZE + 1];
SQLINTEGER sqlcode;
SQLSMALLINT length;

    while ( SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode, buffer,
                    SQL_MAX_MESSAGE_LENGTH + 1, &length) == SQL_SUCCESS )
    {
        printf("\n **** ERROR ****\n");
        printf("          SQLSTATE: %s\n", sqlstate);
        printf("Native Error Code: %ld\n", sqlcode);
        printf("%s \n", buffer);
    };
    return;
}

/*****/
int check_error (SQLHENV   henv,
                SQLHDBC   hdbc,
                SQLHSTMT  hstmt,
                SQLRETURN  frc)
{
SQLRETURN  rc;

    print_error(henv, hdbc, hstmt);

    switch (frc){
case SQL_SUCCESS : break;
case SQL_ERROR :
case SQL_INVALID_HANDLE:
        printf("\n ** FATAL ERROR, Attempting to rollback transaction **\n");
        rc = SQLTransact(henv, hdbc, SQL_ROLLBACK);
        if (rc != SQL_SUCCESS)
            printf("Rollback Failed, Exiting application\n");
        else
            printf("Rollback Successful, Exiting application\n");
        terminate(henv, hdbc);
        exit(frc);
        break;
case SQL_SUCCESS_WITH_INFO :
        printf("\n ** Warning Message, application continuing\n");
        break;
case SQL_NO_DATA_FOUND :
        printf("\n ** No Data Found ** \n");
        break;
default :
        printf("\n ** Invalid Return Code ** \n");
        printf(" ** Attempting to rollback transaction **\n");
        SQLTransact(henv, hdbc, SQL_ROLLBACK);
        terminate(henv, hdbc);
        exit(frc);
        break;
    }
    return(SQL_SUCCESS);
}

```

---

## Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

| SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS  
| PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER  
| EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR  
| CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND  
| NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

| UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR  
| ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

- | 1. LOSS OF, OR DAMAGE TO, DATA;
- | 2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC  
| CONSEQUENTIAL DAMAGES; OR
- | 3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

| SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT,  
| INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS  
| OR EXCLUSIONS MAY NOT APPLY TO YOU.

---

## Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other 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 grant 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 or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country 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 which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

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 information and all licensed material available for it are provided
- | by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement,
- | IBM License Agreement for Machine Code, 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 contains 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 contains 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming Interface Information

This SQL CLI documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- | DB2
- | DB2 Universal Database
- | eServer
- | e(logo) server
- | i5/OS
- | IBM
- | IBM(logo)
- | iSeries
- | OS/390

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

- | Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

---

## 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, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works 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.







Printed in USA