IBM® DB2 Universal Database™

Call Level Interface Guide and Reference, Volume 1

Version 8

SC09-4849-00

IBM® DB2 Universal Database™

Call Level Interface Guide and Reference, Volume 1

Version 8

SC09-4849-00

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

Contents

Part 1. CLI Background Information

Chapter 1. Introduction to CLI

Introduction to CLI

DB2 Call Level Interface (DB2 CLI) is IBM's callable SQL interface to the DB2 family of database servers. It is a 'C' and 'C++' application programming interface for relational database access that uses function calls to pass dynamic SQL statements as function arguments. It is an alternative to embedded dynamic SQL, but unlike embedded SQL, DB2 CLI does not require host variables or a precompiler.

DB2 CLI is based on the Microsoft** Open Database Connectivity** (ODBC) specification, and the International Standard for SQL/CLI. These specifications were chosen as the basis for the DB2 Call Level Interface in an effort to follow industry standards and to provide a shorter learning curve for those application programmers already familiar with either of these database interfaces. In addition, some DB2 specific extensions have been added to help the application programmer specifically exploit DB2 features.

The DB2 CLI driver also acts as an ODBC driver when loaded by an ODBC driver manager. It conforms to ODBC 3.51.

DB2 CLI Background Information:

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

The X/Open Company and the SQL Access Group jointly developed a specification for a callable SQL interface referred to as the *X/Open Call Level Interface*. The goal of this interface is to increase the portability of applications by enabling them to become independent of any one database vendor's programming interface. Most of the X/Open Call Level Interface specification has been accepted as part of the ISO Call Level Interface International Standard (ISO/IEC 9075-3:1995 SQL/CLI).

Microsoft® developed a callable SQL interface called Open Database Connectivity (ODBC) for Microsoft operating systems based on a preliminary draft of X/Open CLI.

The ODBC specification also includes an operating environment where database specific ODBC Drivers are dynamically loaded at run time by a driver manager based on the data source (database name) provided on the connect request. The application is linked directly to a single driver manager library rather than to each DBMS's library. The driver manager mediates the application's function calls at run time and ensures they are directed to the appropriate DBMS specific ODBC driver. Since the ODBC driver manager only knows about the ODBC-specific functions, DBMS-specific functions cannot be accessed in an ODBC environment. DBMS-specific dynamic SQL statements are supported via a mechanism called an escape clause.

ODBC is not limited to Microsoft operating systems; other implementations are available on various platforms.

The DB2 CLI load library can be loaded as an ODBC driver by an ODBC driver manager. For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows® platform, the ODBC SDK is available as part of the Microsoft Data Access Components (MDAC) SDK, available for download from http://www.microsoft.com/data/. For non-Windows platforms, the ODBC SDK is provided by other vendors. When developing ODBC applications that may connect to DB2 servers, use this book (for information on DB2 specific extensions and diagnostic information), in conjunction with the OBDC Programmer's Reference and SDK Guide available from Microsoft.

Applications written directly to DB2 CLI link directly to the DB2 CLI load library. DB2 CLI includes support for many ODBC and ISO SQL/CLI functions, as well as DB2 specific functions.

The following DB2 features are available to both ODBC and DB2 CLI applications:
- double byte (graphic) data types
- stored procedures
- Distributed Unit of Work (DUOW), two phase commit
- compound SQL
- user defined types (UDT)
- user defined functions (UDF)

Related concepts:
- "Comparison of DB2 CLI and Microsoft ODBC" on page 11
- "DB2 Call Level Interface (CLI) versus Embedded Dynamic SQL" on page 5
- "Advantages of DB2 CLI over Embedded SQL" on page 6
- "When to Use DB2 CLI or Embedded SQL" on page 8

DB2 Call Level Interface (CLI) versus Embedded Dynamic SQL

An application that uses an embedded SQL interface requires a precompiler to convert the SQL statements into code, which is then compiled, bound to the database, and executed. In contrast, a DB2 CLI application does not have to be precompiled or bound, but instead uses a standard set of functions to execute SQL statements and related services at run time.

This difference is important because, traditionally, precompilers have been specific to each database product, which effectively ties your applications to that product. DB2 CLI enables you to write portable applications that are independent of any particular database product. This independence means DB2 CLI applications do not have to be recompiled or rebound to access different DB2® databases, including host system databases. They just connect to the appropriate database at run time.

The following are differences and similarities between DB2 CLI and embedded SQL:

- DB2 CLI does not require the explicit declaration of cursors. DB2 CLI has a supply of cursors that get used 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 used in DB2 CLI. Instead, the execution of a `SELECT` automatically causes a cursor to be opened.
- Unlike embedded SQL, DB2 CLI allows the use of parameter markers on the equivalent of the `EXECUTE IMMEDIATE` statement (the `SQLExecDirect()` function).
- A `COMMIT` or `ROLLBACK` in DB2 CLI is typically issued via the `SQLEndTran()` function call rather than by executing it as an SQL statement, however, doing do is permitted.
- DB2 CLI manages statement related information on behalf of the application, and provides an abstract object to represent the information called a *statement handle*. This handle eliminates 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 global variables and connection specific information. The *descriptor handle* describes either the parameters of an SQL statement or the columns of a result set.
- DB2 CLI applications can dynamically describe parameters in an SQL statement the same way that CLI and embedded SQL applications describe result sets. This enables CLI applications to dynamically process SQL statements that contain parameter markers without knowing the data type

of those parameter markers in advance. When the SQL statement is prepared, describe information is returned detailing the data types of the parameters.
- DB2 CLI uses the SQLSTATE values defined by the X/Open SQL CAE specification. Although the format and most of the values are consistent with values used by the IBM® relational database products, there are differences. (There are also differences between ODBC SQLSTATES and the X/Open defined SQLSTATES).

Despite these differences, there is an important common concept between embedded SQL and DB2 CLI: *DB2 CLI can execute any SQL statement that can be prepared dynamically in embedded SQL.*

Note: DB2 CLI can also accept some SQL statements that cannot be prepared dynamically, such as compound SQL statements.

Each DBMS may have additional statements that you can dynamically prepare. In this case, DB2 CLI passes the statements directly to the DBMS. There is one exception: the COMMIT and ROLLBACK statements can be dynamically prepared by some DBMSs but will be intercepted by DB2 CLI and treated as an appropriate `SQLEndTran()` request. However, it is recommeded you use the `SQLEndTran()` function to specify either the COMMIT or ROLLBACK statement.

Related reference:
- "Supported SQL Statements" in the *Application Development Guide: Programming Client Applications*

Advantages of DB2 CLI over Embedded SQL

The DB2 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 the application is connected to.
- It increases the portability of applications by removing the dependence on precompilers. Applications are distributed not as embedded SQL source code which must be preprocessed for each database product, but as compiled applications or run time libraries.
- Individual DB2 CLI applications do not need to be bound to each database, only bind files shipped with DB2 CLI need to be bound once for all DB2 CLI applications. This can significantly reduce the amount of management required for the application once it is in general use.

- DB2 CLI applications can connect to multiple databases, including multiple connections to the same database, all from the same application. Each connection has its own commit scope. This is much simpler using CLI than using embedded SQL where the application must make use of multi-threading to achieve the same result.
- DB2 CLI eliminates the need for application controlled, often complex data areas, such as the SQLDA and SQLCA, typically associated with embedded SQL applications. Instead, DB2 CLI allocates and controls the necessary data structures, and provides a *handle* for the application to reference them.
- DB2 CLI enables the development of multi-threaded thread-safe applications where each thread can have its own connection and a separate commit scope from the rest. DB2 CLI achieves this by eliminating the data areas described above, and associating all such data structures that are accessible to the application with a specific handle. Unlike embedded SQL, a multi-threaded CLI application does not need to call any of the context management DB2® APIs; this is handled by the DB2 CLI driver automatically.
- DB2 CLI provides enhanced parameter input and fetching capability, allowing arrays of data to be specified on input, retrieving multiple rows of a result set directly into an array, and executing statements that generate multiple result sets.
- DB2 CLI provides a consistent interface to query catalog (Tables, Columns, Foreign Keys, Primary Keys, etc.) information contained in the various DBMS catalog tables. The result sets returned are consistent across DBMSs. This shields the application from catalog changes across releases of database servers, as well as catalog differences amongst different database servers; thereby saving applications from writing version specific and server specific catalog queries.
- Extended data conversion is also provided by DB2 CLI, requiring less application code when converting information between various SQL and C data types.
- DB2 CLI incorporates both the ODBC and X/Open CLI functions, both of which are accepted industry specifications. DB2 CLI is also aligned with the ISO CLI standard. Knowledge that application developers invest in these specifications can be applied directly to DB2 CLI development, and vice versa. This interface is intuitive to grasp for those programmers who are familiar with function libraries but know little about product specific methods of embedding SQL statements into a host language.
- DB2 CLI provides the ability to retrieve multiple rows and result sets generated from a stored procedure residing on a DB2 Universal Database (or DB2 Universal Database for OS/390 and z/OS version 5 or later) server. However, note that this capability exists for Version 5 DB2 Universal Database clients using embedded SQL if the stored procedure resides on a server accessible from a DataJoiner® Version 2 server.

- DB2 CLI offers more extensive support for scrollable cursors. With scrollable cursors, you can scroll through a cursor as follows:
  - Forward by one or more rows
  - Backward by one or more rows
  - From the first row by one or more rows
  - From the last row by one or more rows.

  Scrollable cursors can be used in conjunction with array output. You can declare an updateable cursor as scrollable then move forward or backward through the result set by one or more rows. You can also fetch rows by specifying an offset from:
  - The current row
  - The beginning or end of the result set
  - A specific row you have previously set with a bookmark.

When to Use DB2 CLI or Embedded SQL

Which interface you choose depends on your application.

DB2 CLI is ideally suited for query-based graphical user interface (GUI) applications that require portability. The advantages listed above, may make using DB2 CLI seem like the obvious choice for any application. There is however, one factor that must be considered, the comparison between static and dynamic SQL. It is much easier to use static SQL in embedded applications.

Static SQL has several advantages:

- Performance

  Dynamic SQL is prepared at run time, static SQL is prepared at precompile time. As well as requiring more processing, the preparation step may incur additional network-traffic at run time. The additional network traffic can be avoided if the DB2 CLI application makes use of deferred prepare (which is the default behavior).

  It is important to note that static SQL will not always have better performance than dynamic SQL. Dynamic SQL is prepared at runtime and uses the database statistics available at that time, whereas static SQL makes use of database statistics available at BIND time. Dynamic SQL can make use of changes to the database, such as new indexes, to choose the optimal access plan, resulting in potentially better performance than the same SQL executed as static SQL. In addition, precompilation of dynamic SQL statements can be avoided if they are cached.
- Encapsulation and Security

In static SQL, the authorizations to access objects (such as a table, view) are associated with a package and are validated at package binding time. This means that database administrators need only to grant execute on a particular package to a set of users (thus encapsulating their privileges in the package) without having to grant them explicit access to each database object. In dynamic SQL, the authorizations are validated at run time on a per statement basis; therefore, users must be granted explicit access to each database object. This permits these users access to parts of the object that they do not have a need to access.

- Embedded SQL is supported in languages other than C or C++.
- For fixed query selects, embedded SQL is simpler.

If an application requires the advantages of both interfaces, it is possible to make use of static SQL within a DB2 CLI application by creating a stored procedure that contains the static SQL. The stored procedure is called from within a DB2 CLI application and is executed on the server. Once the stored procedure is created, any DB2 CLI or ODBC application can call it.

It is also possible to write a mixed application that uses both DB2 CLI and embedded SQL, taking advantage of their respective benefits. In this case, DB2 CLI is used to provide the base application, with key modules written using static SQL for performance or security reasons. This complicates the application design, and should only be used if stored procedures do not meet the applications requirements.

Ultimately, the decision on when to use each interface, will be based on individual preferences and previous experience rather than on any one factor.

Related concepts:

- “CLI/ODBC/JDBC Trace Facility” on page 223

Related tasks:

- “Preparing and Executing SQL Statements in CLI Applications” on page 31
- “Issuing SQL Statements in CLI Applications” on page 29
- “Creating Static SQL with CLI/ODBC/JDBC Static Profiling” on page 217

Chapter 2. DB2 CLI and ODBC

Comparison of DB2 CLI and Microsoft ODBC

This topic discusses the support provided by the DB2 ODBC driver, and how it differs from DB2 CLI.

Figure 1 on page 12 below compares DB2 CLI and the DB2 ODBC driver. The left side shows an ODBC driver under the ODBC Driver Manager, and the right side illustrates DB2 CLI, the callable interface designed for DB2® specific applications.

DB2 Client refers to all available DB2 Clients. DB2 refers to all DB2 Universal Database products.

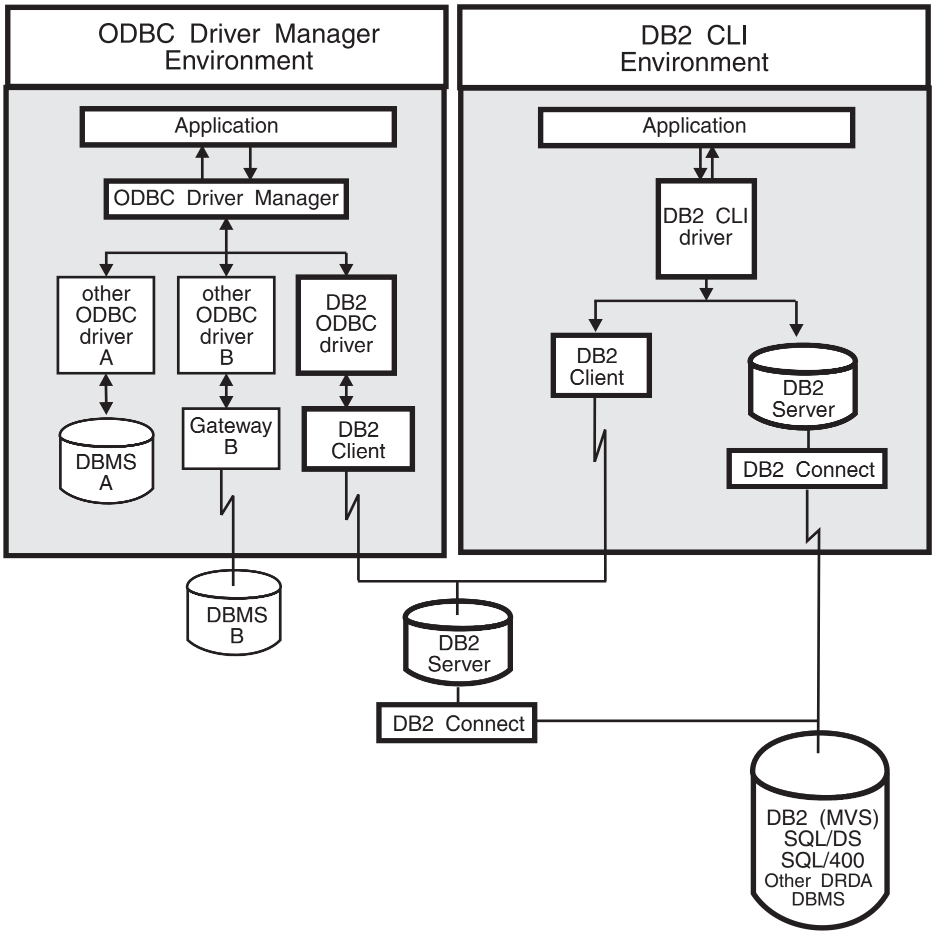


Figure 1. DB2 CLI and ODBC.

In an ODBC environment, the Driver Manager provides the interface to the application. It also dynamically loads the necessary *driver* for the database server that the application connects to. It is the driver that implements the ODBC function set, with the exception of some extended functions implemented by the Driver Manager. In this environment DB2 CLI conforms to ODBC 3.51.

For ODBC application development, you must obtain an ODBC Software Development Kit. For the Windows® platform, the ODBC SDK is available as part of the Microsoft® Data Access Components (MDAC) SDK, available for download from http://www.microsoft.com/data/. For non-Windows platforms, the ODBC SDK is provided by other vendors.

In environments without an ODBC driver manager, DB2 CLI is a self sufficient driver which supports a subset of the functions provided by the ODBC driver. Table 1 summarizes the two levels of support, and the CLI and ODBC function summary provides a complete list of ODBC functions and indicates if they are supported.

Table 1. DB2 CLI ODBC Support

ODBC Features	**DB2 ODBC Driver**	**DB2 CLI**
Core Level Functions	All	All
Level 1 Functions	All	All
Level 2 Functions	All	All, except for `SQLDrivers()`
Additional DB2 CLI Functions	All, functions can be accessed by dynamically loading the DB2 CLI library.	• `SQLSetConnectAttr()` • `SQLGetEnvAttr()` • `SQLSetEnvAttr()` • `SQLSetColAttributes()` • `SQLGetSQLCA()` • `SQLBindFileToCol()` • `SQLBindFileToParam()` • `SQLExtendedBind()` • `SQLExtendedPrepare()` • `SQLGetLength()` • `SQLGetPosition()` • `SQLGetSubString()`

Table 1. DB2 CLI ODBC Support (continued)

ODBC Features	DB2 ODBC Driver	DB2 CLI
SQL Data Types	All the types listed for DB2 CLI.	• SQL_BIGINT • SQL_BINARY • SQL_BIT • SQL_BLOB • SQL_BLOB_LOCATOR • SQL_CHAR • SQL_CLOB • SQL_CLOB_LOCATOR • SQL_DBCLOB • SQL_DBCLOB_LOCATOR • SQL_DECIMAL • SQL_DOUBLE • SQL_FLOAT • SQL_GRAPHIC • SQL_INTEGER • SQL_LONG • SQL_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SHORT • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR

Table 1. DB2 CLI ODBC Support (continued)

ODBC Features	**DB2 ODBC Driver**	**DB2 CLI**
C Data Types	All the types listed for DB2 CLI.	• SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_DATE • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TIME • SQL_C_TIMESTAMP • SQL_C_TINYINT • SQL_C_SBIGINT • SQL_C_UBIGINT • SQL_C_NUMERIC ** • SQL_C_WCHAR ** **Only supported on Windows platform**
Return Codes	All the codes listed for DB2 CLI.	• SQL_SUCCESS • SQL_SUCCESS_WITH_INFO • SQL_STILL_EXECUTING • SQL_NEED_DATA • SQL_NO_DATA_FOUND • SQL_ERROR • SQL_INVALID_HANDLE
SQLSTATES	Mapped to X/Open SQLSTATES with additional IBM® SQLSTATES, with the exception of the ODBC type **08**S01.	Mapped to X/Open SQLSTATES with additional IBM SQLSTATES
Multiple connections per application	Supported	Supported
Dynamic loading of driver	Supported	Not applicable

Isolation Levels:

The following table map IBM RDBMs isolation levels to ODBC transaction isolation levels. The `SQLGetInfo()` function indicates which isolation levels are available.

Table 2. Isolation Levels Under ODBC

IBM Isolation Level	**ODBC Isolation Level**
Cursor Stability	SQL_TXN_READ_COMMITTED
Repeatable Read	SQL_TXN_SERIALIZABLE_READ
Read Stability	SQL_TXN_REPEATABLE_READ
Uncommitted Read	SQL_TXN_READ_UNCOMMITTED
No Commit	(no equivalent in ODBC)
Note: `SQLSetConnectAttr()` and `SQLSetStmtAttr()` will return SQL_ERROR with an SQLSTATE of **HY**009 if you try to set an unsupported isolation level.	

Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*
- "Introduction to CLI" on page 3

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "CLI and ODBC Function Summary" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Part 2. Programming CLI Applications

Chapter 3. Writing a Basic CLI Application

There are three core components of a CLI application: initialization, transaction processing, and termination. This chapter describes the key steps of a typical CLI application.

Intialization

Handles in CLI

A CLI handle is a variable that refers to a data object allocated and managed by DB2 CLI. Using handles relieves the application from having to allocate and manage global variables or data structures, such as the SQLDA.

There are four types of handles in CLI:

Environment handle
An environment handle refers to a data object that holds information about the global state of the application, such as attributes or valid connections. An environment handle must be allocated before a connection handle can be allocated.

Connection handle
A connection handle refers to a data object that holds information

associated with a connection to a particular data source (database). Examples of such information include valid statement and descriptor handles on a connection, transaction status, and diagnostic information.

An application can be connected to several data sources at the same time, and can establish several distinct connections to the same data source. A separate connection handle must be allocated for each concurrent connection. A connection handle must be allocated before a statement or descriptor handle can be allocated.

Connection handles ensure that multithreaded applications which use one connection per thread are thread-safe, because separate data structures are allocated and maintained by DB2 CLI for each connection.

Note: There is a limit of 512 active connections per environment handle.

Statement handle

A statement handle refers to a data object that is used to track the execution of a single SQL statement. It provides access to statement information such as error messages, the associated cursor name, and status information for SQL statement processing. A statement handle must be allocated before an SQL statement can be issued.

When a statement handle is allocated, DB2 CLI automatically allocates four descriptors and assigns the handles for these descriptors to the SQL_ATTR_APP_ROW_DESC, SQL_ATTR_APP_PARAM_DESC, SQL_ATTR_IMP_ROW_DESC, and SQL_ATTR_IMP_PARAM_DESC statement attributes. Application descriptors can be explicitly allocated by allocating descriptor handles.

The number of statement handles available to a CLI application depends on the number of large packages the application has defined and is limited by overall system resources (usually stack size). By default, there are 3 small and 3 large packages. Each small package allows a maximum of 64 statement handles per connection, and each large package allows a maximum of 384 statement handles per connection. The number of available statement handles by default is therefore (3 * 64) + (3 * 384) = 1344.

To get more than the default 1344 statement handles, increase the number of large packages by setting the value of the CLI/ODBC configuration keyword CLIPKG to a value up to 30. CLIPKG indicates the number of large packages that will be generated. If you set CLIPKG to the maximum value of 30, then the maximum number of statement handles that is available becomes (3 * 64) + (30 * 384) = 11 712.

An HY014 SQLSTATE may be returned on the call to `SQLPrepare()`, `SQLExecute()`, or `SQLExecDirect()` if this limit is exceeded.

It is recommended that you only allocate as many large packages as your application needs to run, as packages take up space in the database.

Descriptor handle

A descriptor handle refers to a data object that contains information about the columns in a result set and dynamic parameters in an SQL statement.

On operating systems that support multiple threads, applications can use the same environment, connection, statement, or descriptor handle on different threads. DB2 CLI provides thread safe access for all handles and function calls. The application itself might experience unpredictable behavior if the threads it creates do not co-ordinate their use of DB2 CLI resources.

Related concepts:

- "Descriptors in CLI Applications" on page 177
- "Handle Freeing in CLI Applications" on page 48

Related tasks:

- "Initializing CLI Applications" on page 24
- "Allocating Statement Handles in CLI Applications" on page 28
- "Freeing Statement Resources in CLI Applications" on page 46

Related reference:

- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLPrepare Function (CLI) - Prepare a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLDA (SQL descriptor area)" in the *SQL Reference, Volume 1*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "DB2 CLI Bind Files and Package Names" on page 243
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword" on page 346
- "CLIPKG CLI/ODBC Configuration Keyword" on page 302

Initialization and Termination in CLI Overview

Figure 2 on page 23 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 in CLI Overview.

The initialization task consists of the allocation and initialization of environment and connection handles. An environment handle must be allocated before a connection handle can be created. Once a connection handle is created, the application can then establish a connection. When a connection exists, the application can proceed to the transaction processing task. An application then passes the appropriate handle when it calls other DB2 CLI functions.

The termination task consists of disconnecting from the data source and freeing those handles that were allocated during the initialization phase. The connection handle should be freed before freeing the environment handle.

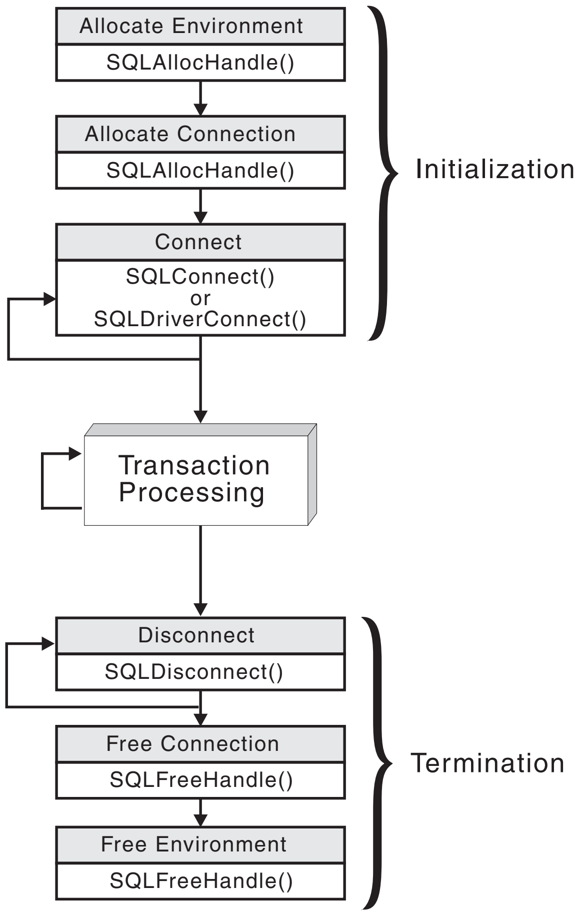


Figure 2. Conceptual View of Initialization and Termination Tasks

Related concepts:

- "Handles in CLI" on page 19
- "Transaction Processing in CLI Overview" on page 26

Related reference:

- "SQLAllocHandle Function (CLI) - Allocate Handle" in the *CLI Guide and Reference, Volume 2*
- "SQLConnect Function (CLI) - Connect to a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLDisconnect Function (CLI) - Disconnect from a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLDriverConnect Function (CLI) - (Expanded) Connect to a Data Source" in the *CLI Guide and Reference, Volume 2*

- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*

Initializing CLI Applications

Initializing CLI applications is part of the larger task of programming with CLI. The task of initializing CLI applications involves allocating environment and connection handles and then connecting to the data source.

Procedure:

To initialize the application:

1. Allocate an environment handle by calling `SQLAllocHandle()` with a *HandleType* of SQL_HANDLE_ENV and an *InputHandle* of SQL_NULL_HANDLE. For example:

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

   Use the allocated environment handle, returned in the *OutputHandlePtr argument (henv in the example above), for all subsequent calls that require an environment handle.
2. Optional: Set environment attributes for your application by calling `SQLSetEnvAttr()` with the desired environment attribute for each attribute you want set.

   Important: If you plan to run your application as an ODBC application, you must set the SQL_ATTR_ODBC_VERSION environment attribute using `SQLSetEnvAttr()`. Setting this attribute for applications that are strictly DB2 CLI applications is recommended but not required.
3. Allocate a connection handle by calling `SQLAllocHandle()` with a *HandleType* of SQL_HANDLE_DBC using the environment handle returned from Step 1 as the *InputHandle* argument. For example:

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

   Use the allocated connection handle, returned in the **OutputHandlePtr* argument (hdbc in the example above), for all subsequent calls that require a connection handle.
4. Optional: Set connection attributes for your application by calling `SQLSetConnectAttr()` with the desired connection attribute for each attribute you want set.
5. Connect to a data source by calling one of following functions with the connection handle you allocated in Step 3 for each data source you want to connect to:
   - SQLConnect(): basic database connection method. For example:

     ```
     SQLConnect (hdbc, server, SQL_NTS, user, SQL_NTS, password, SQL_NTS);
     ```

where SQL_NTS is a special string length value that indicates the referenced string is null-terminated.

- SQLDriverConnect(): extended connect function that allows additional connect options and offers Graphical User Interface support. For example:

```
char * connStr = "DSN=SAMPLE;UID=;PWD=;";

SQLDriverConnect (hdbc, (SQLHWND)NULL, connStr, SQL_NTS,
                  NULL, 0, NULL, SQL_DRIVER_NOPROMPT);
```

- SQLBrowseConnect(): least common connection method that iteratively returns the attributes and attribute values for connecting to a data source. For example:

```
char * connInStr = "DSN=SAMPLE;UID=;PWD=;";
char outStr[512];

SQLBrowseConnect (hdbc, connInStr, SQL_NTS, outStr,
                  512, &strLen2Ptr);
```

Now that your application has been initialized, you can proceed to processing transactions.

Related concepts:

- "Handles in CLI" on page 19
- "Transaction Processing in CLI Overview" on page 26

Related reference:

- "SQLAllocHandle Function (CLI) - Allocate Handle" in the *CLI Guide and Reference, Volume 2*
- "SQLBrowseConnect Function (CLI) - Get Required Attributes to Connect to Data source" in the *CLI Guide and Reference, Volume 2*
- "SQLConnect Function (CLI) - Connect to a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLDriverConnect Function (CLI) - (Expanded) Connect to a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*
- "SQLSetEnvAttr Function (CLI) - Set Environment Attribute" in the *CLI Guide and Reference, Volume 2*
- "Environment Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "clihandl.c -- How to allocate and free handles (CLI)"
- "dbcongui.c -- How to connect to a database with a graphical user interface (GUI) (CLI)"

- “dbconn.c -- How to connect to and disconnect from a database (CLI)”

Transaction Processing

Transaction Processing in CLI Overview

Figure 3 on page 27 shows the typical order of function calls in the transaction processing task of a DB2 CLI application. Not all functions or possible paths are shown.

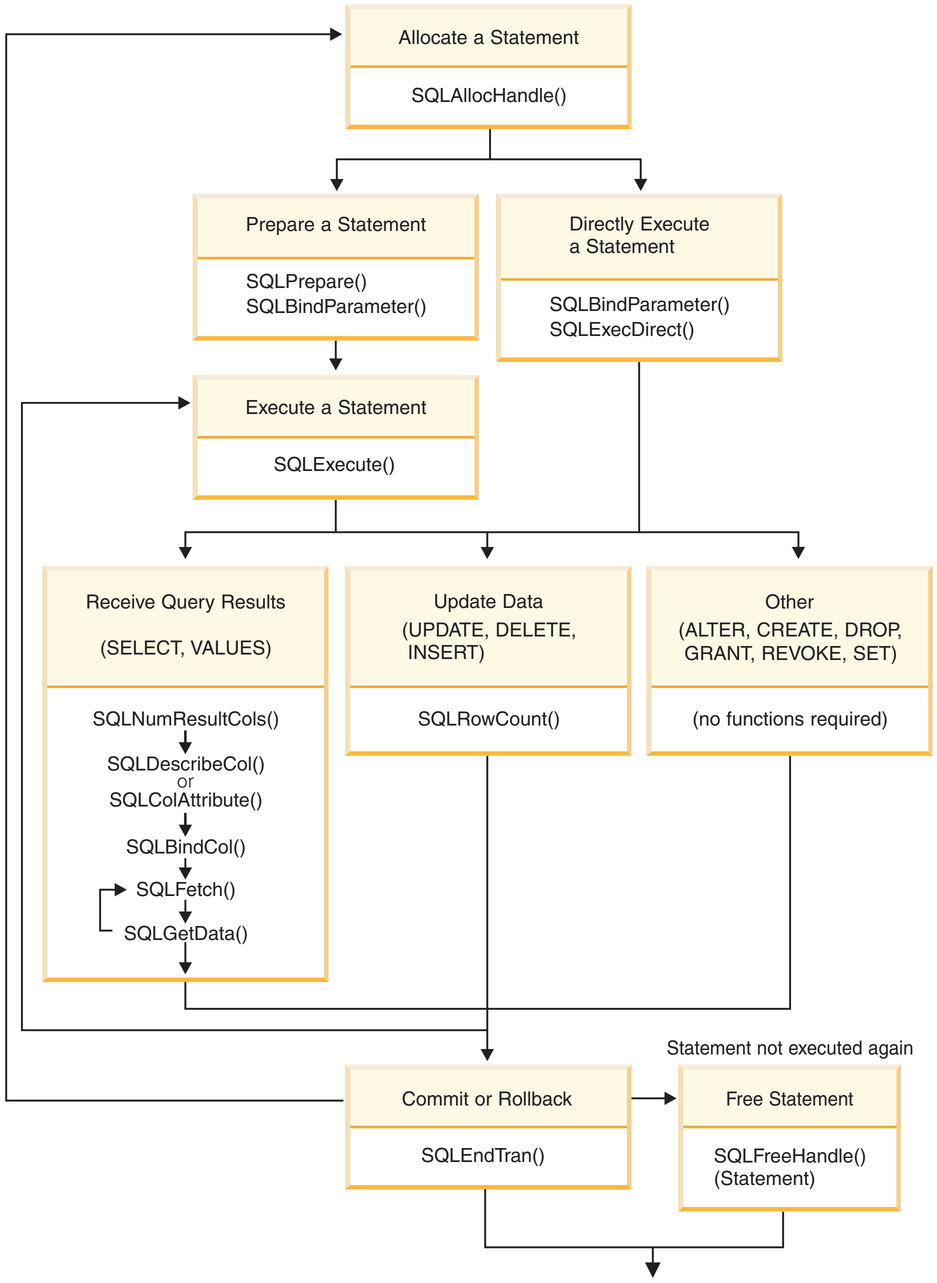


Figure 3. Transaction Processing

The transaction processing task contains five steps:
- Allocating statement handle(s)
- Preparing and executing SQL statements

- Processing results
- Committing or Rolling Back
- (Optional) Freeing statement handle(s) if the statement is unlikely to be executed again.

Related concepts:
- "Commit Modes in CLI Applications" on page 38

Related tasks:
- "Allocating Statement Handles in CLI Applications" on page 28
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Retrieving Query Results in CLI Applications" on page 41
- "Updating and Deleting Data in CLI Applications" on page 45
- "Freeing Statement Resources in CLI Applications" on page 46
- "Terminating a CLI Application" on page 64

Allocating Statement Handles in CLI Applications

To issue an SQL statement in a CLI application, you need to allocate a statement handle. A statement handle tracks the execution of a single SQL statement and is associated with a connection handle. Allocating statement handles is part of the larger task of processing transactions.

Prerequisites:

Before you begin allocating statement handles, you must allocate an environment handle and a connection handle. This is part of the task of initializing your CLI application.

Procedure:

To allocate a statement handle:
1. Call `SQLAllocHandle()` with a *HandleType* of SQL_HANDLE_STMT. For example:
   ```
   SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmt);
   ```
2. Optional: To set attributes for this statement, call `SQLSetStmtAttr()` for each desired attribute option.

After allocating environment, connection, and statement handles, you can now prepare, issue, or execute SQL statements.

Related concepts:
- "Transaction Processing in CLI Overview" on page 26

Related tasks:

- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Issuing SQL Statements in CLI Applications" on page 29

Related reference:

- "SQLAllocHandle Function (CLI) - Allocate Handle" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "clihandl.c -- How to allocate and free handles (CLI)"

Issuing SQL Statements in CLI Applications

SQL statements are passed to DB2 CLI functions as SQLCHAR string variables. The variable can consist of one or more SQL statements, with or without parameter markers, depending on the type of processing you want. This topic describes the various ways SQL statements can be issued in DB2 CLI applications.

Before you issue an SQL statement, ensure you have allocated a statement handle.

Procedure:

Perform either of the following steps to issue SQL statements:

- To issue a single SQL statement, either initialize an SQLCHAR variable with the SQL statement and pass this variable to the CLI function, or directly pass a string argument cast to an SQLCHAR * to the function. For example:

  ```
  SQLCHAR * stmt = (SQLCHAR *) "SELECT deptname, location FROM org";
  /* ... */
  SQLExecDirect (hstmt, stmt, SQL_NTS);
  ```

  or

  ```
  SQLExecDirect (hstmt, (SQLCHAR *) "SELECT deptname, location FROM org",
                 SQL_NTS);
  ```

- To issue multiple SQL statements on the same statement handle, either initialize an array of SQLCHAR elements, where each element represents an individual SQL statement, or initialize a single SQLCHAR variable that contains the multiple statements delimited by a ";" character. For example:

```
SQLCHAR * multiple_stmts[] = {
  (SQLCHAR *) "SELECT deptname, location FROM org",
  (SQLCHAR *) "SELECT id, name FROM staff WHERE years > 5",
  (SQLCHAR *) "INSERT INTO org VALUES (99,'Hudson',20,'Western','Seattle')"
};
```

or

```
SQLCHAR * multiple_stmts =
  "SELECT deptname, location FROM org;
   SELECT id, name FROM staff WHERE years > 5;
   INSERT INTO org VALUES (99, 'Hudson', 20, 'Western', 'Seattle')";
```

Note: When a list of SQL statements is specified, only one statement is executed at a time, starting with the first statement in the list. Each subsequent statement is executed in the order it appears. (To execute subsequent statements, you must call `SQLMoreResults()`.)

- To issue SQL statements with parameter markers, see Binding Parameter Markers.
- To capture and convert SQL statements dynamically executed with DB2 CLI (dynamic SQL) to static SQL, see Creating Static SQL.

Related concepts:

- "Parameter Marker Binding in CLI Applications" on page 33

Related tasks:

- "Binding Parameter Markers in CLI Applications" on page 36
- "Creating Static SQL with CLI/ODBC/JDBC Static Profiling" on page 217
- "Allocating Statement Handles in CLI Applications" on page 28

Related reference:

- "C Data Types for CLI Applications" on page 54
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLMoreResults Function (CLI) - Determine If There Are More Result Sets" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "dbuse.c -- How to use a database (CLI)"
- "tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)"

Preparing and Executing SQL Statements in CLI Applications

Once you have allocated a statement handle, you can then perform operations using SQL statements. An SQL statement must be prepared before it can be executed, and DB2 CLI offers two ways of preparing and executing an SQL statement:

- perform the prepare and execute operations in separate steps
- combine the prepare and execute operations into one step

Prerequisites:

Before preparing and executing your SQL statement, ensure you have allocated a statement handle for it.

Procedure:

To prepare and execute an SQL statement in separate steps:

1. Prepare the SQL statement by calling `SQLPrepare()` and passing the SQL statement as the *StatementText* argument.
2. Call `SQLBindParameter()` to bind any parameter markers you may have in the SQL statement.
3. Execute the prepared statement by calling `SQLExecute()`.

Use this method when:

- The same SQL statement will be executed repeatedly (usually with different parameter values). This avoids having to prepare the same statement more than once. The subsequent executions make use of the access plans already generated by the prepare, thus increasing driver efficiency and delivering better application performance.
- The application requires information about the parameters or columns in the result set prior to statement execution.

To prepare and execute an SQL statement in one step:

1. Call `SQLBindParameter()` to bind any parameter markers you may have in the SQL statement.
2. Prepare and execute the statement by calling `SQLExecDirect()` with the SQL statement as the *StatementText* argument.
3. Optional: If a list of SQL statements are to be executed, call `SQLMoreResults()` to advance to the next SQL statement.

Use this method of preparing and executing in one step when:

- The statement will be executed only once. This avoids having to call two functions to execute the statement.

- The application does not require information about the columns in the result set before the statement is executed.

Related concepts:
- "Deferred Prepare in CLI Applications" on page 32
- "Transaction Processing in CLI Overview" on page 26

Related tasks:
- "Allocating Statement Handles in CLI Applications" on page 28

Related reference:
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLMoreResults Function (CLI) - Determine If There Are More Result Sets" in the *CLI Guide and Reference, Volume 2*
- "SQLPrepare Function (CLI) - Prepare a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*

Related samples:
- "dbuse.c -- How to use a database (CLI)"
- "tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)"

Deferred Prepare in CLI Applications

Deferred prepare is the name of the CLI feature that seeks to minimise communication with the server by sending both the prepare and execute requests for SQL statements in the same network flow. The default value for this property can be overridden using the CLI/ODBC configuration keyword DEFERREDPREPARE. This property can be set on a per-statement handle basis by calling `SQLSetStmtAttr()` to change the SQL_ATTR_DEFEFERRED_PREPARE statement attribute.

When deferred prepare is on, the prepare request is not sent to the server until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance. Because of this behavior, any errors that would typically be generated by `SQLPrepare()` will appear at execute time, and `SQLPrepare()` will always return SQL_SUCCESS. Deferred prepare is of greatest benefit when the application generates queries where the answer set is very small, and the overhead of separate requests and replies is not spread across multiple blocks of query data.

Note: Even if deferred prepare is enabled, operations that require a statement to be prepared prior to the operation's execution will force the prepare request to be sent to the server before the execute. Describe operations resulting from calls to `SQLDescribeParam()` or `SQLDescribeCol()` are examples of when deferred prepare will be overridden, because describe information is only available after the statement has been prepared.

Related tasks:

- "Preparing and Executing SQL Statements in CLI Applications" on page 31

Related reference:

- "SQLDescribeCol Function (CLI) - Return a Set of Attributes for a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLDescribeParam Function (CLI) - Return Description of a Parameter Marker" in the *CLI Guide and Reference, Volume 2*
- "SQLPrepare Function (CLI) - Prepare a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60
- "DEFERREDPREPARE CLI/ODBC Configuration Keyword" on page 316

Parameter Marker Binding in CLI Applications

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 executed. (A parameter marker is used where a host variable would be used in static embedded SQL.) This value can be obtained from:

- An application variable.

  `SQLBindParameter()` is used to bind the application storage area to the parameter marker.
- A LOB value from the database server (by specifying a LOB locator).

  `SQLBindParameter()` is used to bind a LOB locator to the parameter marker. The LOB value itself is supplied by the database server, so only the LOB locator is transferred between the database server and the application.
- A file within the application's environment containing a LOB value.

`SQLBindFileToParam()` is used to bind a file to a LOB parameter marker. When `SQLExecDirect()` is executed, DB2 CLI will transfer the contents of the file directly to the database server.

Parameter markers are referenced sequentially, from left to right, starting at 1. `SQLNumParams()` can be used to determine the number of parameters in a statement.

The application must bind an application variable to each parameter marker in the SQL statement before it executes that statement. Binding is carried out by calling the `SQLBindParameter()` function with a number of arguments to indicate:

- the ordinal position of the parameter,
- the SQL type of the parameter,
- the type of parameter (input, output, or inout),
- the C data type of the variable,
- a pointer to the application variable,
- the length of the variable.

The bound application variable and its associated length are called *deferred* input arguments because only the pointers are passed when the parameter is bound; no data is read from the variable until the statement is executed. Deferred arguments allow the application to modify the contents of the bound parameter variables, and re-execute the statement with the new values.

Information about each parameter remains in effect until:

- it is overridden by the application
- the application unbinds the parameter by calling `SQLFreeStmt()` with the SQL_RESET_PARAMS *Option*
- the application drops the statement handle by calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_STMT or `SQLFreeStmt()` with the SQL_DROP *Option*.

Information for each parameter remains in effect until overridden, or until the application unbinds the parameter or drops the statement handle. If the application executes the SQL statement repeatedly without changing the parameter binding, then DB2 CLI uses the same pointers to locate the data on each execution. The application can also change the parameter binding to a different set of deferred variables by calling `SQLBindParameter()` again for one or more parameters and specifying different application variables. The application must not deallocate or discard variables used for deferred input fields between the time it binds the fields to parameter markers and the time

DB2 CLI accesses them at execution time. Doing so can result in DB2 CLI reading garbage data, or accessing invalid memory resulting in an application trap.

It is possible to bind the parameter to a variable of a different type from that required by the SQL statement. The application must indicate the C data type of the source, and the SQL type of the parameter marker, and DB2 CLI will convert the contents of the variable to match the SQL data type specified. For example, the SQL statement may require an integer value, but your application has a string representation of an integer. The string can be bound to the parameter, and DB2 CLI will convert the string to the corresponding integer value when you execute the statement.

By default, DB2 CLI does not verify the type of the parameter marker. If the application indicates an incorrect type for the parameter marker, it could cause:
- an extra conversion by the DBMS
- an error at the DBMS which forces DB2 CLI to describe the statement being executed and re-execute it, resulting in extra network traffic
- an error returned to the application if the statement cannot be described, or the statement cannot be re-executed successfully.

Information about the parameter markers can be accessed using descriptors. If you enable automatic population of the implementation parameter descriptor (IPD) then information about the parameter markers will be collected. The statement attribute SQL_ATTR_ENABLE_AUTO_IPD must be set to SQL_TRUE for this to work.

If the parameter marker is part of a predicate on a query and is associated with a User Defined Type, then the parameter marker must be cast to the built-in type in the predicate portion of the statement; otherwise, an error will occur.

After the SQL statement has been executed, and the results processed, the application may wish to reuse the statement handle to execute 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.

Related concepts:
- "Handles in CLI" on page 19
- "Data Types and Data Conversion in CLI Applications" on page 49
- "Large Object Usage in CLI Applications" on page 116
- "Descriptors in CLI Applications" on page 177

- "User Defined Type (UDT) Usage in CLI Applications" on page 174

Related reference:

- "SQLBindFileToParam Function (CLI) - Bind LOB File Reference to LOB Parameter" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeStmt Function (CLI) - Free (or Reset) a Statement Handle" in the *CLI Guide and Reference, Volume 2*
- "SQLNumParams Function (CLI) - Get Number of Parameters in A SQL Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "dbuse.c -- How to use a database (CLI)"
- "dtlob.c -- How to read and write LOB data (CLI)"
- "spclient.c -- Call various stored procedures (CLI)"
- "tbmod.c -- How to modify table data (CLI)"
- "tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)"

Binding Parameter Markers in CLI Applications

This topic describes how to bind parameter markers to application variables before executing SQL statements. Parameter markers in SQL statements can be bound to single values or to arrays of values. Binding each parameter marker individually requires a network flow to the server for each set of values. Using arrays, however, allows several sets of parameter values to be bound and sent at once to the server.

Prerequisites:

Before you bind parameter markers, ensure you have initialized your application.

Procedure:

To bind parameter markers, perform either of the following steps:

- To bind parameter markers one at a time to application variables, call `SQLBindParameter()` for each application variable you want to bind. Ensure you specify the correct parameter type: SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT. The following example shows how two parameter markers are bound with two application variables:

```
SQLCHAR *stmt =
   (SQLCHAR *)"DELETE FROM org WHERE deptnumb = ? AND division = ? ";
SQLSMALLINT parameter1 = 0;
char parameter2[20];

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

/* bind parameter2 to the statement */
cliRC = SQLBindParameter(hstmt,
                         2,
                         SQL_PARAM_INPUT,
                         SQL_C_CHAR,
                         SQL_VARCHAR,
                         20,
                         0,
                         parameter2,
                         20,
                         NULL);
```

- To bind at once many values to parameter markers, perform either of the following tasks which use arrays of values:
  - binding parameter markers with column-wise array input
  - binding parameter markers with row-wise array input

Related concepts:

- "Parameter Marker Binding in CLI Applications" on page 33

Related tasks:

- "Binding Parameter Markers in CLI Applications with Column-Wise Array Input" on page 95
- "Binding Parameter Markers in CLI Applications with Row-Wise Array Input" on page 96
- "Initializing CLI Applications" on page 24

Related reference:

- “SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator” in the *CLI Guide and Reference, Volume 2*

Related samples:
- “dbuse.c -- How to use a database (CLI)”
- “tbmod.c -- How to modify table data (CLI)”
- “tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)”

Commit Modes in CLI Applications

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 guaranteed to be completed (committed) or undone (rolled back), as if they were a single operation. When the transaction spans multiple connections, it is referred to as a distributed unit of work (DUOW).

Transactions are started implicitly with the first access to the database using `SQLPrepare()`, `SQLExecDirect()`, `SQLGetTypeInfo()`, or any function that returns a result set, such as catalog functions. At this point a transaction has begun, even if the call failed.

DB2 CLI supports two commit modes:

auto-commit
: In auto-commit mode, every SQL statement is a complete transaction, which is automatically committed. For a non-query statement, the commit is issued at the end of statement execution. For a query statement, the commit is issued after the cursor has been closed. The default commit mode is auto-commit (except when participating in a coordinated transaction).

manual-commit
: In manual-commit mode, the transaction ends when you use `SQLEndTran()` to either rollback or commit the transaction. This means that any statements executed (on the same connection) between the start of a transaction and the call to `SQLEndTran()` are treated as a single transaction. If DB2 CLI is in manual-commit mode, a new transaction is implicitly started when an SQL statement that can be contained within a transaction is executed against the current data source.

An application can switch between manual-commit and auto-commit modes by calling `SQLSetConnectAttr()`. Auto-commit can be useful for query-only applications, because the commits can be chained to the SQL execution request sent to the server. Another benefit of auto-commit is improved concurrency since locks are removed as soon as possible. Applications that need to perform updates to the database should turn off auto-commit as soon as the database connection has been established and should not wait until the disconnect before committing or rolling back the transaction.

The following are examples of how to set auto-commit on and off:

- Setting auto-commit on:

```
/* ... */

/* set AUTOCOMMIT on */
sqlrc = SQLSetConnectAttr( hdbc,
                           SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)SQL_AUTOCOMMIT_ON, SQL_NTS) ;

/* continue with SQL statement execution */
```

- Setting auto-commit off:

```
/* ... */

/* set AUTOCOMMIT OFF */
sqlrc = SQLSetConnectAttr( hdbc,
                           SQL_ATTR_AUTOCOMMIT,
                           (SQLPOINTER)SQL_AUTOCOMMIT_OFF, SQL_NTS) ;

/* ... */

/* execute the statement */
/* ... */
sqlrc = SQLExecDirect( hstmt, stmt, SQL_NTS ) ;

/* ... */

sqlrc = SQLEndTran( SQL_HANDLE_DBC, hdbc, SQL_ROLLBACK );
DBC_HANDLE_CHECK( hdbc, sqlrc);

/* ... */
```

When multiple connections exist to the same or different databases, each connection has its own transaction. Special care must be taken to call `SQLEndTran()` with the correct connection handle to ensure that only the intended connection and related transaction is affected. It is also possible to rollback or commit all the connections by specifying a valid environment handle, and a NULL connection handle on the `SQLEndTran()` call. Unlike distributed unit of work connections, there is no coordination between the transactions on each connection in this case.

Related concepts:

- "Catalog Functions for Querying System Catalog Information in CLI Applications" on page 195
- "Cursors in CLI Applications" on page 75
- "DB2 as Transaction Manager in CLI Applications" on page 158

- "Multisite Updates (Two Phase Commit) in CLI Applications" on page 157

Related tasks:

- "Updating and Deleting Data in CLI Applications" on page 45

Related reference:

- "SQLEndTran Function (CLI) - End Transactions of a Connection or an Environment" in the *CLI Guide and Reference, Volume 2*
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLGetTypeInfo Function (CLI) - Get Data Type Information" in the *CLI Guide and Reference, Volume 2*
- "SQLPrepare Function (CLI) - Prepare a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "tut_mod.c -- How to modify table data (CLI)"
- "tut_read.c -- How to read data from tables (CLI)"

When to Call the CLI SQLEndTran() Function

In auto-commit mode, a commit is issued implicitly at the end of each statement execution or when a cursor is closed.

In manual-commit mode, `SQLEndTran()` must be called before calling `SQLDisconnect()`. If a Distributed Unit of Work is involved, additional rules may apply.

Consider the following when deciding where in the application to end a transaction:

- Each connection cannot have more than one current transaction at any given time, so keep dependent statements within the same unit of work. Note that statements must always be kept on the same connection under which they were allocated.
- Various resources may be held while the current transaction on a connection is running. Ending the transaction will release the resources for use by other applications.
- Once a transaction has successfully been committed or rolled back, it is fully recoverable from the system logs. Open transactions are not recoverable.

Effects of Calling SQLEndTran():

When a transaction ends:
- All locks on DBMS objects are released, except those that are associated with a held cursor.
- Prepared statements are preserved from one transaction to the next. Once a statement has been prepared on a specific statement handle, it does not need to be prepared again even after a commit or rollback, provided the statement continues to be associated with the same statement handle.
- Cursor names, bound parameters, and column bindings are maintained from one transaction to the next.
- By default, cursors are preserved after a commit (but not a rollback). All cursors are by default defined with the WITH HOLD clause, except when the CLI application is running in a Distributed Unit of Work environment.

Related concepts:
- "Handles in CLI" on page 19
- "Parameter Marker Binding in CLI Applications" on page 33
- "Commit Modes in CLI Applications" on page 38
- "Cursors in CLI Applications" on page 75
- "Multisite Updates (Two Phase Commit) in CLI Applications" on page 157
- "Column Binding in CLI Applications" on page 100

Related reference:
- "SQLDisconnect Function (CLI) - Disconnect from a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLEndTran Function (CLI) - End Transactions of a Connection or an Environment" in the *CLI Guide and Reference, Volume 2*

Related samples:
- "dbmcon.c -- How to use multiple databases (CLI)"
- "dbuse.c -- How to use a database (CLI)"
- "tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)"

Retrieving Query Results in CLI Applications

Retrieving query results is part of the larger task of processing transactions in CLI applications. Retrieving query results involves binding application variables to columns of a result set and then fetching the rows of data into the application variables. A typical query is the SELECT statement.

Prerequisites:

Before you retrieve results, ensure you have initialized your application and prepared and executed the necessary SQL statements.

Procedure:

To retrieve each row of the result set:

1. Optional: Determine the structure of the result set, number of columns, and column types and lengths by calling `SQLNumResultCols()` and `SQLDescribeCol()`.

   Note: Performing this step can reduce performance if done before the query has been executed, because it forces CLI to describe the query's columns. Information about the result set's columns is available after successful execution, and describing the result set does not incur any additional overhead if the describe is performed after successful execution.
2. Bind an application variable to each column of the result set, by calling `SQLBindCol()`, ensuring that the variable type matches the column type. For example:

   ```
   struct
   {
     SQLINTEGER ind;
     SQLSMALLINT val;
   }
   deptnumb; /* variable to be bound to the DEPTNUMB column */

   struct
   {
     SQLINTEGER ind;
     SQLCHAR val[15];
   }
   location; /* variable to be bound to the LOCATION column */

   /* ... */

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

   /* bind column 2 to variable */
   cliRC = SQLBindCol(hstmt, 2, SQL_C_CHAR, location.val, 15,
                      &location.ind);
   STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);
   ```

   The application can use the information obtained in step 1 to determine an appropriate C data type for the application variable and to allocate the maximum storage the column value could occupy. The columns are bound to deferred output arguments, which means the data is written to these storage locations when it is fetched.

Important: Do not de-allocate or discard variables used for deferred output arguments between the time the application binds them to columns of the result set and the time DB2 CLI writes to these arguments.

3. Repeatedly fetch the row of data from the result set by calling SQLFetch() until SQL_NO_DATA_FOUND is returned. For example:

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

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

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

   SQLFetchScroll() can also be used to fetch multiple rows of the result set into an array.

   If data conversion was required for the data types specified on the call to SQLBindCol(), the conversion will occur when SQLFetch() is called.

4. Optional: Retrieve columns that were not previously bound by calling SQLGetData() after each successful fetch. You can retrieve all unbound columns this way. For example:

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

   if (cliRC == SQL_NO_DATA_FOUND)
   {
     printf("\n  Data not found.\n");
   }
   while (cliRC != SQL_NO_DATA_FOUND)
   {
     /* use SQLGetData() to get the results */
     /* get data from column 1 */
     cliRC = SQLGetData(hstmt,
                        1,
                        SQL_C_SHORT,
                        &deptnumb.val,
                        0,
                        &deptnumb.ind);
     STMT_HANDLE_CHECK(hstmt, hdbc, cliRC);

     /* get data from column 2 */
     cliRC = SQLGetData(hstmt,
                        2,
   ```

```
                         SQL_C_CHAR,
                         location.val,
                         15,
                         &location.ind);

    /* display the data */
    printf("    %-8d %-14.14s \n", deptnumb.val, location.val);

    /* fetch the next row */
    cliRC = SQLFetch(hstmt);
  }
```

Note: Applications perform better if columns are bound, rather than having them retrieved as unbound columns using `SQLGetData()`. However, an application may be constrained in the amount of long data it can retrieve and handle at one time. If this is a concern, then `SQLGetData()` may be the better choice.

Related concepts:

- "Data Types and Data Conversion in CLI Applications" on page 49

Related tasks:

- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Retrieving Array Data in CLI Applications Using Column-wise Binding" on page 104
- "Retrieving Array Data in CLI Applications Using Row-wise Binding" on page 106

Related reference:

- "C Data Types for CLI Applications" on page 54
- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLDescribeCol Function (CLI) - Return a Set of Attributes for a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLNumResultCols Function (CLI) - Get Number of Result Columns" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Related samples:

- “tbread.c -- How to read data from tables (CLI)”
- “tut_read.c -- How to read data from tables (CLI)”

Updating and Deleting Data in CLI Applications

Updating and deleting data is part of the larger task of processing transactions in CLI. There are two types of update and delete operations available in CLI programming: simple and positioned. A simple update or delete operation only requires that you issue and execute the UPDATE or DELETE SQL statements as you would any other SQL statement. You could, in this case, use `SQLRowCount()` to obtain the number of rows affected by the SQL statement.

Positioned updates and deletes involve modifying the data of a result set. A positioned update is the update of a column of a result set, and a positioned delete is when a row of a result set is deleted. Positioned update and delete operations require cursors to be used. This document describes how to perform positioned update and delete operations by first getting the name of the cursor associated with the result set, and then issuing and executing the UPDATE or DELETE on a second statement handle using the retrieved cursor name.

Prerequisites:

Before you perform a positioned update or delete operation, ensure that you have initialized your CLI application.

Procedure:

To perform a positioned update or delete operation:

1. Generate the result set that the update or delete will be performed on by issuing and executing the SELECT SQL statement.
2. Call `SQLGetCursorName()` to get the name of the cursor, using the same statement handle as the handle that executed the SELECT statement. This cursor name will be needed in the UPDATE or DELETE statement.

   When a statement handle is allocated, a cursor name is automatically generated. You can define your own cursor name using `SQLSetCursorName()`, but it is recommended that you use the name that is generated by default because all error messages will reference the generated name, not the name defined using `SQLSetCursorName()`.
3. Allocate a second statement handle that will be used to execute the positioned update or delete.

   To update a row that has been fetched, the application uses two statement handles, one for the fetch and one for the update. You cannot reuse the fetch statement handle to execute the positioned update or delete, because it is still in use when the positioned update or delete is executing.

4. Fetch data from the result set by calling `SQLFetch()` or `SQLFetchScroll()`.
5. Issue the UPDATE or DELETE SQL statement with the WHERE CURRENT of clause and specify the cursor name obtained in step 2. For example:
```
sprintf((char *)stmtPositionedUpdate,
        "UPDATE org SET location = 'Toronto' WHERE CURRENT of %s",
        cursorName);
```
6. Position the cursor on the row of the data fetched and execute the positioned update or delete statement.

Related tasks:

- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Issuing SQL Statements in CLI Applications" on page 29

Related reference:

- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLGetCursorName Function (CLI) - Get Cursor Name" in the *CLI Guide and Reference, Volume 2*
- "SQLRowCount Function (CLI) - Get Row Count" in the *CLI Guide and Reference, Volume 2*
- "DELETE statement" in the *SQL Reference, Volume 2*
- "UPDATE statement" in the *SQL Reference, Volume 2*

Related samples:

- "spserver.c -- Definition of various types of stored procedures (CLI)"
- "tbmod.c -- How to modify table data (CLI)"

Freeing Statement Resources in CLI Applications

After a transaction has completed, end the processing for each statement handle by freeing the resources associated with it. There are four main tasks that are involved with freeing resources for a statement handle:

- close the open cursor
- unbind the column bindings
- unbind the parameter bindings
- free the statement handle

There are two ways you can free statement resources: using `SQLFreeHandle()` or `SQLFreeStmt()`.

Prerequisites:

Before you can free statement resources, you must have initialized your CLI application and allocated a statement handle.

Procedure:

To free statement resources with `SQLFreeHandle()`, call `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_STMT and the handle you want to free. This will close any open cursor associated with this statement handle, unbind column and parameter bindings, and free the statement handle. This invalidates the statement handle. You do not need to explicitly carry out each of the four tasks listed above.

To free statement resources with `SQLFreeStmt()`, you need to call `SQLFreeStmt()` for each task (depending on how the application was implemented, all of these tasks may not be necessary):

- To close the open cursor, call `SQLCloseCursor()`, or call `SQLFreeStmt()` with the SQL_CLOSE *Option* and statement handle as arguments. This closes the cursor and discards any pending results.
- To unbind column bindings, call `SQLFreeStmt()` with an *Option* of SQL_UNBIND and the statement handle. This unbinds all columns for this statement handle except the bookmark column.
- To unbind parameter bindings, call `SQLFreeStmt()` with an *Option* of SQL_RESET_PARAMS and the statement handle. This releases all parameter bindings for this statement handle.
- To free the statement handle, call `SQLFreeStmt()` with an *Option* of SQL_DROP and the statement handle to be freed. This invalidates this statement handle.

  Note: Although this option is still supported, we recommend that you use `SQLFreeHandle()` in your DB2 CLI applications so that they conform to the latest standards.

Related concepts:

- "Handles in CLI" on page 19
- "Handle Freeing in CLI Applications" on page 48

Related tasks:

- "Initializing CLI Applications" on page 24
- "Allocating Statement Handles in CLI Applications" on page 28

Related reference:

- "SQLCloseCursor Function (CLI) - Close Cursor and Discard Pending Results" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeStmt Function (CLI) - Free (or Reset) a Statement Handle" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "tut_read.c -- How to read data from tables (CLI)"
- "tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)"
- "utilcli.c -- Utility functions used by DB2 CLI samples (CLI)"

Handle Freeing in CLI Applications

Environment Handle:

Prior to calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_ENV, an application must call `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DBC for all connections allocated under the environment. Otherwise, the call to `SQLFreeHandle()` returns SQL_ERROR and the environment remains valid, as well as any connection associated with that environment.

Connection Handle:

If a connection is open on the handle, an application must call `SQLDisconnect()` for the connection prior to calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DBC. Otherwise, the call to `SQLFreeHandle()` returns SQL_ERROR and the connection remains valid.

Statement Handle:

A call to `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_STMT frees all resources that were allocated by a call to `SQLAllocHandle()` with a *HandleType* of SQL_HANDLE_STMT. When an application calls `SQLFreeHandle()` to free a statement that has pending results, the pending results are discarded. When an application frees a statement handle, DB2 CLI frees all the automatically generated descriptors associated with that handle.

Note that `SQLDisconnect()` automatically drops any statements and descriptors open on the connection.

Descriptor Handle:

A call to `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DESC frees the descriptor handle in *Handle*. The call to `SQLFreeHandle()` does not release any memory allocated by the application that may be referenced by the deferred fields (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR) of any descriptor record of *Handle*. When an explicitly allocated descriptor handle is freed, all statements that the freed handle had been associated with revert to their automatically allocated descriptor handle.

Note that `SQLDisconnect()` automatically drops any statements and descriptors open on the connection. When an application frees a statement handle, DB2 CLI frees all the automatically generated descriptors associated with that handle.

Related concepts:

- "Descriptors in CLI Applications" on page 177

Related tasks:

- "Freeing Statement Resources in CLI Applications" on page 46
- "Terminating a CLI Application" on page 64

Related reference:

- "SQLDisconnect Function (CLI) - Disconnect from a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*
- "Descriptor Header and Record Field Initialization Values (CLI)" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Data Types and Data Conversion in CLI Applications

When writing a DB2 CLI application it is necessary to work with both SQL data types and C data types. This is unavoidable because the DBMS uses SQL data types, while the application uses C data types. The application, therefore, must match C data types to SQL data types when calling DB2 CLI functions to transfer data between the DBMS and the application.

To facilitate this, DB2 CLI provides symbolic names for the various data types, and manages the transfer of data between the DBMS and the application. It also performs data conversion (from a C character string to an SQL INTEGER type, for example) if required. DB2 CLI needs to know both the source and target data type. This requires the application to identify both data types using symbolic names.

Data type conversion can occur under one of two conditions:
- The application specified a C type that is not the default C type for the SQL type.
- The application specified an SQL type that does not match the base column SQL type at the server, and there was no describe information available to the DB2 CLI driver.

Example of How to Use Data Types:

Because the data source contains SQL data types and the CLI application works with C data types, the data to be retrieved needs to be handled with the correct data types. The following example shows how SQL and C data types are used by an application to retrieve data from the source into application variables. The example is taken from the `tut_read.c` sample program and examines how data is retrieved from the DEPTNUMB column of the ORG table in the sample database.
- The DEPTNUMB column of the ORG table is declared as the SQL data type SMALLINT.
- The application variable which will hold the retrieved data is declared using C types. Since the DEPTNUMB column is of SQL type SMALLINT, the application variable needs to be declared using the C type SQLSMALLINT, which is equivalent to the SQL type SMALLINT.

  ```
  struct
  {   SQLINTEGER ind;
      SQLSMALLINT val;
  } deptnumb;         /* variable to be bound to the DEPTNUMB column */
  ```

  SQLSMALLINT represents the base C type of short int.
- The application binds the application variable to the symbolic C data type of SQL_C_SHORT:

  ```
  sqlrc = SQLBindCol(hstmt, 1, SQL_C_SHORT, &deptnumb.val, 0,
                     &deptnumb.ind);
  ```

  The data types are now consistent, because the result data type SQL_C_SHORT represents the C type SQLSMALLINT.

Data Conversion:

DB2 CLI manages the transfer and any required conversion of data between the application and the DBMS. Before the data transfer actually takes place, either the source, the target or both data types are indicated when calling `SQLBindParameter()`, `SQLBindCol()` or `SQLGetData()`. These functions use the symbolic type names to identify the data types involved.

For example, to bind a parameter marker that corresponds to an SQL data type of DECIMAL(5,3), to an application's C buffer type of double, the appropriate `SQLBindParameter()` call would look like:

```
SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_DOUBLE,
                  SQL_DECIMAL, 5, 3, double_ptr, 0, NULL);
```

The functions mentioned in the previous paragraph can be used to convert data from the default to other data types, but not all data conversions are supported or make sense.

The rules that specify limits on precision and scale, as well as truncation and rounding rules for type conversions apply in DB2 CLI, with the following exception: truncation of values to the right of the decimal point for numeric values may return a truncation warning, whereas truncation to the left of the decimal point returns an error. In cases of error, the application should call `SQLGetDiagRec()` to obtain the SQLSTATE and additional information on the failure. When moving and converting floating point data values between the application and DB2 CLI, no correspondence is guaranteed to be exact as the values may change in precision and scale.

Related concepts:

- "SQLSTATES for DB2 CLI" on page 62

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "Data Conversions Supported in CLI" on page 357
- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagRec Function (CLI) - Get Multiple Fields Settings of Diagnostic Record" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "dtinfo.c -- How get information about data types (CLI)"
- "tut_read.c -- How to read data from tables (CLI)"

SQL Symbolic and Default Data Types for CLI Applications

The table below lists each of the SQL data types used by CLI applications, with its corresponding symbolic name, and the default C symbolic name.

SQL Data Type
: This column contains the SQL data types as they would appear in an SQL CREATE statement. The SQL data types are dependent on the DBMS.

Symbolic SQL Data Type
: This column contains SQL symbolic names that are defined (in `sqlcli.h`) as an integer value. These values are used by various functions to identify the SQL data types listed in the first column.

Default C Symbolic Data Type
: This column contains C symbolic names, also defined as integer values. These values are used in various function arguments to identify the C data type. The symbolic names are used by various functions, such as `SQLBindParameter()`, `SQLGetData()`, and `SQLBindCol()` to indicate the C data types of the application variables. Instead of explicitly identifying C data types when calling these functions, SQL_C_DEFAULT can be specified instead, and DB2 CLI will assume a default C data type based on the SQL data type of the parameter or column as shown by this table. For example, the default C data type of SQL_DECIMAL is SQL_C_CHAR.

  It is recommended that applications do not use `SQL_C_DEFAULT` to define C data types because it is less efficient for the CLI driver. Explicitly indicating the C data type in the application is preferred since it yields better performance than using `SQL_C_DEFAULT`.

Table 3. SQL Symbolic and Default Data Types

SQL Data Type	Symbolic SQL Data Type	Default Symbolic C Data Type
BIGINT	SQL_BIGINT	SQL_C_BIGINT
BLOB	SQL_BLOB	SQL_C_BINARY
BLOB LOCATOR [a]	SQL_BLOB_LOCATOR	SQL_C_BLOB_LOCATOR
CHAR	SQL_CHAR	SQL_C_CHAR
CHAR	SQL_TINYINT	SQL_C_TINYINT
CHAR FOR BIT DATA [b]	SQL_BINARY	SQL_C_BINARY
CHAR FOR BIT DATA	SQL_BIT	SQL_C_BIT
CLOB	SQL_CLOB	SQL_C_CHAR
CLOB LOCATOR [a]	SQL_CLOB_LOCATOR	SQL_C_CLOB_LOCATOR
DATALINK	SQL_DATALINK	SQL_C_CHAR
DATE	SQL_TYPE_DATE	SQL_C_TYPE_DATE
DBCLOB	SQL_DBCLOB	SQL_C_DBCHAR
DBCLOB LOCATOR [a]	SQL_DBCLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR

Table 3. SQL Symbolic and Default Data Types (continued)

SQL Data Type	Symbolic SQL Data Type	Default Symbolic C Data Type
DECIMAL	SQL_DECIMAL	SQL_C_CHAR
DOUBLE	SQL_DOUBLE	SQL_C_DOUBLE
FLOAT	SQL_FLOAT	SQL_C_DOUBLE
GRAPHIC	SQL_GRAPHIC	SQL_C_DBCHAR
INTEGER	SQL_INTEGER	SQL_C_LONG
LONG VARCHAR [b]	SQL_LONGVARCHAR	SQL_C_CHAR
LONG VARCHAR FOR BIT DATA [b]	SQL_LONGVARBINARY	SQL_C_BINARY
LONG VARGRAPHIC [b]	SQL_LONGVARGRAPHIC	SQL_C_DBCHAR
LONG VARGRAPHIC [b]	SQL_WLONGVARCHAR	SQL_C_DBCHAR
NUMERIC [c]	SQL_NUMERIC [c]	SQL_C_CHAR
REAL	SQL_REAL	SQL_C_FLOAT
SMALLINT	SQL_SMALLINT	SQL_C_SHORT
TIME	SQL_TYPE_TIME	SQL_C_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP
VARCHAR	SQL_VARCHAR	SQL_C_CHAR
VARCHAR FOR BIT DATA [b]	SQL_VARBINARY	SQL_C_BINARY
VARGRAPHIC	SQL_VARGRAPHIC	SQL_C_DBCHAR
VARGRAPHIC	SQL_WVARCHAR	SQL_C_DBCHAR
WCHAR	SQL_WCHAR	SQL_C_WCHAR

a LOB locator types are not persistent SQL data types, (columns can not be defined with a locator type, they are only used to describe parameter markers, or to represent a LOB value).
b LONG data types and FOR BIT DATA data types should be replaced by an appropriate LOB types whenever possible.
c NUMERIC is a synonym for DECIMAL on DB2 for z/OS, DB2 for VSE & VM and DB2 Universal Database.

Note: The data types DATE, DECIMAL, NUMERIC, TIME, and TIMESTAMP cannot be transferred to their default C buffer types without a conversion.

Related concepts:

- "Data Types and Data Conversion in CLI Applications" on page 49
- "LOB Locators in CLI Applications" on page 118

Related reference:

- "C Data Types for CLI Applications" on page 54
- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*

C Data Types for CLI Applications

The following table lists the generic type definitions for each symbolic C type that is used in CLI applications.

C Symbolic Data Type
This column contains C symbolic names, defined as integer values. These values are used in various function arguments to identify the C data type shown in the last column.

C Type
This column contains C defined types, defined in `sqlcli.h` using a C `typedef` statement. The values in this column should be used to declare all DB2 CLI related variables and arguments, in order to make the application more portable. Refer to Table 6 on page 56 for a list of additional symbolic data types used for function arguments.

Base C type
This column is shown for reference only. All variables and arguments should be defined using the symbolic types in the previous column. Some of the values are C structures that are described in Table 5 on page 55.

Table 4. C Data Types

C Symbolic Data Type	C Type	Base C type
SQL_C_BINARY	SQLCHAR	unsigned char
SQL_C_BIT	SQLCHAR	unsigned char or char (Value 1 or 0)
SQL_C_BLOB_LOCATOR [a]	SQLINTEGER	long int
SQL_C_CLOB_LOCATOR [a]	SQLINTEGER	long int
SQL_C_CHAR	SQLCHAR	unsigned char
SQL_C_DATALINK	SQLCHAR	unsigned char
SQL_C_DBCHAR	SQLDBCHAR	wchar_t
SQL_C_DBCLOB_LOCATOR	SQLINTEGER	long int
SQL_C_DOUBLE	SQLDOUBLE	double

Table 4. C Data Types (continued)

C Symbolic Data Type	C Type	Base C type
SQL_C_FLOAT	SQLREAL	float
SQL_C_LONG	SQLINTEGER	long int
SQL_C_NUMERIC [b]	SQL_NUMERIC_STRUCT	see Table 5
SQL_C_SBIGINT	SQLBIGINT	_int64
SQL_C_SHORT	SQLSMALLINT	short int
SQL_C_TINYINT	SQLSCHAR	signed char (Range -128 to 127)
SQL_C_TYPE_DATE	DATE_STRUCT	see Table 5
SQL_C_TYPE_TIME	TIME_STRUCT	see Table 5
SQL_C_TYPE_TIMESTAMP	TIMESTAMP_STRUCT	see Table 5
SQL_C_UBIGINT	SQLUBIGINT	unsigned _int64
SQL_C_ULONG	SQLUINTEGER	unsigned long int
SQL_C_USHORT	SQLUSMALLINT	unsigned short int
SQL_C_UTINYINT	SQLUCHAR	unsigned char
SQL_C_WCHAR	SQLWCHAR	wchar_t

- **a** LOB Locator Types.
- **b** Windows only.

Note: SQL file reference data types (used in embedded SQL) are not needed in DB2 CLI.

Table 5. C Structures

C Type	Generic Structure	Windows Structure

DATE_STRUCT	`typedef struct DATE_STRUCT`
	`  {`
	`    SQLSMALLINT   year;`
	`    SQLUSMALLINT   month;`
	`    SQLUSMALLINT   day;`
	`  } DATE_STRUCT;`
		`typedef struct tagDATE_STRUCT`
		`  {`
		`    SWORD   year;`
		`    UWORD   month;`
		`    UWORD   day;`
		`  } DATE_STRUCT;`

TIME_STRUCT	`typedef struct TIME_STRUCT`
	`  {`
	`    SQLUSMALLINT   hour;`
	`    SQLUSMALLINT   minute;`
	`    SQLUSMALLINT   second;`
	`  } TIME_STRUCT;`
		`typedef struct tagTIME_STRUCT`
		`  {`
		`    UWORD   hour;`
		`    UWORD   minute;`
		`    UWORD   second;`
		`  } TIME_STRUCT;`

Table 5. C Structures (continued)

C Type	Generic Structure	Windows Structure
TIMESTAMP_STRUCT	`typedef struct TIMESTAMP_STRUCT { SQLUSMALLINT year; SQLUSMALLINT month; SQLUSMALLINT day; SQLUSMALLINT hour; SQLUSMALLINT minute; SQLUSMALLINT second; SQLINTEGER fraction; } TIMESTAMP_STRUCT;`	`typedef struct tagTIMESTAMP_STRUCT { SWORD year; UWORD month; UWORD day; UWORD hour; UWORD minute; UWORD second; UDWORD fraction; } TIMESTAMP_STRUCT;`
SQL_NUMERIC_STRUCT	(No generic structure. Only a Windows structure.)	`typedef struct tagSQL_NUMERIC_STRUCT { SQLCHAR precision; SQLCHAR scale; SQLCHAR sign;` [a] `SQLCHAR val[SQL_MAX_NUMERIC_LEN];` [b c] `} SQL_NUMERIC_STRUCT;`

Refer to Table 6 for more information on the SQLUSMALLINT C data type.

a Sign field: 1 = positive, 2 = negative

b A number is stored in the val field of the SQL_NUMERIC_STRUCT structure as a scaled integer, in little endian mode (the leftmost byte being the least-significant byte). For example, the number 10.001 base 10, with a scale of 4, is scaled to an integer of 100010. Because this is 186AA in hexadecimal format, the value in SQL_NUMERIC_STRUCT would be "AA 86 01 00 00 ... 00", with the number of bytes defined by the SQL_MAX_NUMERIC_LEN #define.

c The precision and scale fields of the SQL_C_NUMERIC data type are never used for input from an application, only for output from the driver to the application. When the driver writes a numeric value into the SQL_NUMERIC_STRUCT, it will use its own default as the value for the precision field, and it will use the value in the SQL_DESC_SCALE field of the application descriptor (which defaults to 0) for the scale field. An application can provide its own values for precision and scale by setting the SQL_DESC_PRECISION and SQL_DESC_SCALE fields of the application descriptor.

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. Both the generic and ODBC data types are shown below.

Table 6. C Data Types and Base C Data Types

Defined C Type	Base C Type	Typical Usage
SQLPOINTER	void *	Pointer to storage for data and parameters.
SQLHANDLE	long int	Handle used to reference all 4 types of handle information.
SQLHENV	long int	Handle referencing environment information.

Table 6. C Data Types and Base C Data Types (continued)

Defined C Type	Base C Type	Typical Usage
SQLHDBC	long int	Handle referencing database connection information.
SQLHSTMT	long int	Handle referencing statement information.
SQLUSMALLINT	unsigned short int	Function input argument for unsigned short integer values.
SQLUINTEGER	unsigned long int	Function input argument for unsigned long integer values.
SQLRETURN	short int	Return code from DB2 CLI functions.

Related concepts:

- "Data Types and Data Conversion in CLI Applications" on page 49
- "LOB Locators in CLI Applications" on page 118

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51

String Handling in CLI Applications

The following conventions deal with the various aspects of string arguments in DB2 CLI functions.

Length of String Arguments:

Input strings can have an associated length argument which indicates either the exact length of the string (not including the null terminator), the special value SQL_NTS to indicate a null-terminated string, or SQL_NULL_DATA to pass a NULL value. If the length is set to SQL_NTS, DB2 CLI will determine the length of the string by locating the null terminator.

Output strings have two associated length arguments: an input length argument to specify the length of the allocated output buffer, and an output length argument to return the actual length of the string returned by DB2 CLI. The returned length value is the total length of the string available for return, regardless of whether it fits in the buffer or not.

For SQL column data, if the output is a null value, SQL_NULL_DATA is returned in the length argument and the output buffer is untouched. The descriptor field SQL_DESC_INDICATOR_PTR is set to SQL_NULL_DATA if the column value is a null value. For more information, including which other fields are set, see the descriptor FieldIdentifier argument values.

If a function is called with a null pointer for an output length argument, DB2 CLI will not return a length. When the output data is a NULL value, DB2 CLI cannot indicate that the value is NULL. If it is possible that a column in a result set can contain a NULL value, a valid pointer to the output length argument must always be provided. It is highly recommended that a valid output length argument always be used.

Performance Hint:

If the length argument (*StrLen_or_IndPtr*) and the output buffer (*TargetValuePtr*) are contiguous in memory, DB2 CLI can return both values more efficiently, improving application performance. For example, if the following structure is defined:

```
struct
{   SQLINTEGER pcbValue;
    SQLCHAR    rgbValue [BUFFER_SIZE];
} buffer;
```

and `&buffer.pcbValue` and `buffer.rgbValue` is passed to `SQLBindCol()`, DB2 CLI would update both values in one operation.

Null-Termination of Strings:

By default, every character string that DB2 CLI returns is terminated with a null terminator (hex 00), except for strings returned from graphic and DBCLOB data types into SQL_C_CHAR application variables. Graphic and DBCLOB data types that are retrieved into SQL_C_DBCHAR application variables are null terminated with a double byte null terminator. Also, string data retrieved into SQL_C_WCHAR are terminated with the Unicode null terminator 0x0000. This requires that all buffers allocate enough space for the maximum number of bytes expected, plus the null terminator.

It is also possible to use `SQLSetEnvAttr()` and set an environment attribute to disable null termination of variable length output (character string) data. In this case, the application allocates a buffer exactly as long as the longest string it expects. The application must provide a valid pointer to storage for the output length argument so that DB2 CLI can indicate the actual length of data returned; otherwise, the application will not have any means to determine this. The DB2 CLI default is to always write the null terminator.

It is possible, using the PATCH1 CLI/ODBC configuration keyword, to force DB2 CLI to null terminate graphic and DBCLOB strings.

String Truncation:

If an output string does not fit into a buffer, DB2 CLI will truncate the string to the size of the buffer, and write the null terminator. If truncation occurs, the function will return SQL_SUCCESS_WITH_INFO and an SQLSTATE of **01**004 indicating truncation. The application can then compare the buffer length to the output length to determine which string was truncated.

For example, if `SQLFetch()` returns SQL_SUCCESS_WITH_INFO, and an SQLSTATE of **01**004, it means 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 was truncated. You can also call `SQLGetDiagField()` to find out which column failed.

Interpretation of Strings:

Normally, DB2 CLI interprets string arguments in a case-sensitive manner and does not trim any spaces from the values. The one exception is the cursor name input argument on the `SQLSetCursorName()` function: if the cursor name is not delimited (enclosed by double quotes) the leading and trailing blanks are removed and case is ignored.

Related concepts:
- "SQLSTATES for DB2 CLI" on page 62

Related reference:
- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagField Function (CLI) - Get a Field of Diagnostic Data" in the *CLI Guide and Reference, Volume 2*
- "SQLSetEnvAttr Function (CLI) - Set Environment Attribute" in the *CLI Guide and Reference, Volume 2*
- "Descriptor Header and Record Field Initialization Values (CLI)" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60
- "PATCH1 CLI/ODBC Configuration Keyword" on page 325

Diagnostics in CLI Applications Overview

Diagnostics refers to dealing with warning or error conditions generated within an application. There are two levels of diagnostics returned when calling DB2 CLI functions:
- return codes

- detailed diagnostics (SQLSTATEs, messages, SQLCA)

Each CLI function returns the function return code as a basic diagnostic. Both `SQLGetDiagRec()` and `SQLGetDiagField()` provide more detailed diagnostic information. If the diagnostic originates at the DBMS, the `SQLGetSQLCA()` function provides access to the SQLCA. This arrangement lets applications handle the basic flow control based on return codes, and use the SQLSTATES along with the SQLCA to determine the specific causes of failure and to perform specific error handling.

Both `SQLGetDiagRec()` and `SQLGetDiagField()` return three pieces of information:

- SQLSTATE
- Native error: if the diagnostic is detected by the data source, this is the SQLCODE; otherwise, this is set to -99999.
- Message text: this is the message text associated with the SQLSTATE.

`SQLGetSQLCA()` returns the SQLCA for access to specific fields, but should only be used when `SQLGetDiagRec()` or `SQLGetDiagField()` cannot provide the desired information.

Related concepts:

- "SQLSTATES for DB2 CLI" on page 62

Related reference:

- "SQLGetDescRec Function (CLI) - Get Multiple Field Settings of Descriptor Record" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagField Function (CLI) - Get a Field of Diagnostic Data" in the *CLI Guide and Reference, Volume 2*
- "SQLGetSQLCA Function (CLI) - Get SQLCA Data Structure" in the *CLI Guide and Reference, Volume 2*
- "SQLCA (SQL communications area)" in the *SQL Reference, Volume 1*
- "CLI Function Return Codes" on page 60

CLI Function Return Codes

The following table lists all possible return codes for DB2 CLI functions.

Table 7. DB2 CLI Function Return Codes

Return Code	Explanation
SQL_SUCCESS	The function completed successfully, no additional SQLSTATE information is available.

Table 7. DB2 CLI Function Return Codes (continued)

Return Code	Explanation
SQL_SUCCESS_WITH_INFO	The function completed successfully with a warning or other information. Call `SQLGetDiagRec()` or `SQLGetDiagField()` to receive the SQLSTATE and any other informational messages or warnings. The SQLSTATE will have a class of '01'.
SQL_STILL_EXECUTING	The function is running asynchronously and has not yet completed. The DB2 CLI driver has returned control to the application after calling the function, but the function has not yet finished executing.
SQL_NO_DATA_FOUND	The function returned successfully, but no relevant data was found. When this is returned after the execution of an SQL statement, additional information may be available and can be obtained by calling `SQLGetDiagRec()` or `SQLGetDiagField()`.
SQL_NEED_DATA	The application tried to execute an SQL statement but DB2 CLI lacks parameter data that the application had indicated would be passed at execute time.
SQL_ERROR	The function failed. Call `SQLGetDiagRec()` or `SQLGetDiagField()` to receive the SQLSTATE and any other error information.
SQL_INVALID_HANDLE	The function failed due to an invalid input handle (environment, connection or statement handle). This is a programming error. No further information is available.

The following code segment from `tut_read.c` shows how a function return code, SQL_NO_DATA_FOUND, can be used to control when data retrieval should stop:

```
while (cliRC != SQL_NO_DATA_FOUND)
{
  printf("    %-8d %-14.14s \n", deptnumb.val, location.val);

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

Related concepts:

- "Handles in CLI" on page 19
- "Diagnostics in CLI Applications Overview" on page 59

Related reference:
- "CLI and ODBC Function Summary" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagField Function (CLI) - Get a Field of Diagnostic Data" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagRec Function (CLI) - Get Multiple Fields Settings of Diagnostic Record" in the *CLI Guide and Reference, Volume 2*

Related samples:
- "tut_read.c -- How to read data from tables (CLI)"

SQLSTATES for DB2 CLI

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 DB2 CLI or ODBC driver.
- 'IM', is generated by the ODBC driver manager.

Note: Versions of DB2 CLI before Version 5 returned SQLSTATEs with a class of 'S1' rather than 'HY'. To force the CLI driver to return 'S1' SQLSTATEs, the application should set the environment attribute SQL_ATTR_ODBC_VERSION to the value SQL_OV_ODBC2.

DB2 CLI SQLSTATEs include both additional IBM® defined SQLSTATEs that are returned by the database server, and DB2 CLI defined SQLSTATEs for conditions that are not defined in the ODBC version 3 and ISO SQL/CLI specifications. This allows for the maximum amount of diagnostic information to be returned. When running applications in an ODBC environment, 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 `SQLGetDiagRec()` to determine if diagnostic information is available.
- Use the SQLSTATEs rather than the native error code.
- To increase your application's portability, only build dependencies on the subset of DB2 CLI SQLSTATEs that are defined by the ODBC version 3 and ISO SQL/CLI specifications, and return the additional ones as information only. A dependency in an application is a logic flow decision based on specific SQLSTATEs.

  Note: It may be useful to build dependencies on the class (the first 2 characters) of the SQLSTATEs.

- For maximum diagnostic information, return the text message along with the SQLSTATE (if applicable, the text message will also include the IBM defined SQLSTATE). It is also useful for the application to print out the name of the function that returned the error.

The following code segment from `utilcli.c` shows how diagnostic information, such as SQLSTATEs, can be retrieved and displayed:

```
void HandleDiagnosticsPrint(SQLSMALLINT htype, /* handle type identifier */
                            SQLHANDLE hndl /* handle */ )
{
  SQLCHAR message[SQL_MAX_MESSAGE_LENGTH + 1];
  SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
  SQLINTEGER sqlcode;
  SQLSMALLINT length, i;

  i = 1;

  /* get multiple field settings of diagnostic record */
  while (SQLGetDiagRec(htype,
                       hndl,
                       i,
                       sqlstate,
                       &sqlcode,
                       message,
                       SQL_MAX_MESSAGE_LENGTH + 1,
                       &length) == SQL_SUCCESS)
  {
    printf("\n  SQLSTATE          = %s\n", sqlstate);
    printf("  Native Error Code = %ld\n", sqlcode);
    printf("%s\n", message);
    i++;
  }

  printf("-------------------------\n");
}
```

You can use the CLI/ODBC trace facility to gain a better understanding of how your application calls DB2, including any errors that may occur.

Related concepts:

- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:

- "SQLGetDiagRec Function (CLI) - Get Multiple Fields Settings of Diagnostic Record" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Related samples:

- "utilcli.c -- Utility functions used by DB2 CLI samples (CLI)"

Termination

Terminating a CLI Application

After you have initialized your CLI application and processed transactions, you must terminate the application to properly disconnect from the data source and free resources.

Prerequisites:

Before terminating your application, you should have initialized your CLI application and completed processing of all transactions.

Procedure:

To terminate a CLI application:

1. Disconnect from the data source by calling `SQLDisconnect()`.
2. Free the connection handle by calling `SQLFreeHandle()` with a *HandleType* argument of SQL_HANDLE_DBC.

   If multiple database connections exist, repeat steps 1 - 2 until all connections are closed and connection handles freed.
3. Free the environment handle by calling `SQLFreeHandle()` with a *HandleType* argument of SQL_HANDLE_ENV.

Related concepts:

- "Transaction Processing in CLI Overview" on page 26

Related tasks:

- "Initializing CLI Applications" on page 24
- "Freeing Statement Resources in CLI Applications" on page 46

Related reference:

- "SQLDisconnect Function (CLI) - Disconnect from a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "dbconn.c -- How to connect to and disconnect from a database (CLI)"
- "dbmcon.c -- How to use multiple databases (CLI)"
- "utilcli.c -- Utility functions used by DB2 CLI samples (CLI)"

Chapter 4. Programming Hints and Tips

Programming Hints and Tips for CLI Applications

This topic discusses the following subjects:

Common Connection Attributes:

The following connection attributes may need to be set by DB2 CLI applications:

- SQL_ATTR_AUTOCOMMIT - Generally this attribute should be set to SQL_AUTOCOMMIT_OFF, since each commit request can generate extra network flow. Only leave SQL_AUTOCOMMIT_ON on if specifically needed.

  Note: The default is SQL_AUTOCOMMIT_ON.
- SQL_ATTR_TXN_ISOLATION - This connection attribute determines the isolation level at which the connection or statement will operate. The isolation level determines the level of concurrency possible, and the level of

locking required to execute the statement. Applications need to choose an isolation level that maximizes concurrency, yet ensures data consistency.

Common Statement Attributes:

The following statement attributes may need to be set by DB2 CLI applications:

- SQL_ATTR_MAX_ROWS - Setting this attribute limits the number of rows returned to the application from query operations. This can be used to avoid overwhelming an application with a very large result set generated inadvertently, which is especially useful for applications on clients with limited memory resources.

  Setting SQL_ATTR_MAX_ROWS while connected to DB2® for z/OS™ and OS/390 Version 7 and later will add "OPTIMIZE FOR n ROWS" and "FETCH n ROWS ONLY" clauses to the statement. For versions of DB2 for OS/390 prior to Version 7 and any DBMS that does not support the "FETCH n ROWS ONLY" clause, the full result set is still generated at the server using the "OPTIMIZE FOR n ROWS" clause, however DB2 CLI will count the rows on the client and only fetch up to SQL_ATTR_MAX_ROWS rows.
- SQL_ATTR_CURSOR_HOLD - This statement attribute determines if the cursor for this statement will be declared by DB2 CLI using the WITH HOLD clause.

  Resources associated with statement handles can be better utilized by the server if the statements that do not require cursor-hold behavior have this attribute set to SQL_CURSOR_HOLD_OFF. The efficiency gains obtained by the proper use of this attribute are considerable on OS/390 and z/OS.

  Note: Many ODBC applications expect a default behavior where the cursor position is maintained after a commit.
- SQL_ATTR_TXN_ISOLATION - DB2 CLI allows the isolation level to be set at the statement level, however, we recommend that the isolation level be set at the connection level. The isolation level determines the level of concurrency possible, and the level of locking required to execute the statement.

  Resources associated with statement handles can be better utilized by DB2 CLI if statements are set to the required isolation level, rather than leaving all statements at the default isolation level. This should only be attempted with a thorough understanding of the locking and isolation levels of the connected DBMS.

  Applications should use the minimum isolation level possible to maximize concurrency.

Reusing Statement Handles:

Each time a CLI application declares a statement handle, the DB2 CLI driver allocates and then initializes an underlying data structure for that handle. To increase performance, CLI applications can reuse statement handles with different statements, thereby avoiding the costs associated with statement handle allocation and initialization.

Note: Before reusing statement handles, memory buffers and other resources used by the previous statement may need to be released by calling the `SQLFreeStmt()` function. Also, statement attributes previously set on a statement handle (for example, SQL_ATTR_PARAMSET_SIZE) need to be explicitly reset, otherwise they may be inherited by all future statements using the statement handle.

Binding and SQLGetData():

Generally it is more efficient to bind application variables or file references to result sets than to use `SQLGetData()`. When the data is in a LOB column, LOB functions are preferable to `SQLGetData()` (see “Large Object Data Optimization” on page 68 for more information). Use `SQLGetData()` when the data value is large variable-length data that:
- must be received in pieces, or
- may not need to be retrieved.

Limiting Use of Catalog Functions:

Catalog functions, such as `SQLTables()`, force the DB2 CLI driver to query the DBMS catalog tables for information. The queries issued are complex and the DBMS catalog tables can be very large. In general, try to limit the number of times the catalog functions are called, and limit the number of rows returned.

The number of catalog function calls can be reduced by calling the function once, and having the application store (cache) the data.

The number of rows returned can be limited by specifying a:
- Schema name or pattern for all catalog functions
- Table name or pattern for all catalog functions other than `SQLTables()`
- Column name or pattern for catalog functions that return detailed column information.

Remember that although an application may be developed and tested against a data source with hundreds of tables, it may be run against a database with thousands of tables. Consider this likelihood when developing applications.

Close any open cursors (call `SQLCloseCursor()` or `SQLFreeStmt()` with SQL_CLOSE *Option*) for statement handles used for catalog queries to release any locks against the catalog tables. Outstanding locks on the catalog tables can prevent CREATE, DROP or ALTER statements from executing.

Column Names of Function Generated Result Sets:

The column names of the result sets generated by catalog and information functions may change as the ODBC and CLI standards evolve. The *position* of the columns, however, will not change.

Any application dependency should be based on the column position (*iCol* parameter used in `SQLBindCol()`, `SQLGetData()`, and `SQLDescribeCol()`) and not the name.

DB2 CLI-specific Functions Loaded from ODBC Applications:

The ODBC Driver Manager maintains its own set of statement handles which it maps to the CLI statement handles on each call. When a DB2 CLI function is called directly, it must be passed to the CLI driver statement handle, as the CLI driver does not have access to the ODBC mapping.

Call `SQLGetInfo()` with the SQL_DRIVER_HSTMT option to obtain the DB2 CLI statement handle (HSTMT). The DB2 CLI functions can then be called directly from the shared library or DLL, passing the HSTMT argument where required.

Global Dynamic Statement Caching:

DB2 Universal Database servers at version 5 or later for UNIX® or Windows® have a *global dynamic statement cache*. This cache is used to store the most popular access plans for prepared dynamic SQL statements.

Before each statement is prepared, the server automatically searches this cache to see if an access plan has already been created for this exact SQL statement (by this application or any other application or client). If so, the server does not need to generate a new access plan, but will use the one in the cache instead. There is now no need for the application to cache connections at the client unless connecting to a server that does not have a global dynamic statement cache.

Data Insertion and Retrieval Optimization:

The methods that describe using arrays to bind parameters and retrieve data use compound SQL to optimize network flow. Use these methods as much as possible.

Large Object Data Optimization:

Use LOB data types and the supporting functions for long strings whenever possible. Unlike LONG VARCHAR, LONG VARBINARY, and LONG VARGRAPHIC types, LOB data values can use LOB locators and functions such as `SQLGetPostion()` and `SQLGetSubString()` to manipulate large data values at the server.

LOB values can also be fetched directly to a file, and LOB parameter values can be read directly from a file. This saves the overhead of the application transferring data via application buffers.

Case Sensitivity of Object Identifiers:

All database object identifiers, such as table names, view names and column names are stored in the catalog tables in upper case unless the identifier is delimited. If an identifier is created using a delimited name, the exact case of the name is stored in the catalog tables.

When an identifier is referenced within an SQL statement, it is treated as case *insensitive* unless it is delimited.

For example, if the following two tables are created,

```
CREATE TABLE MyTable   (id INTEGER)
CREATE TABLE "YourTable" (id INTEGER)
```

two tables will exist, `MYTABLE` and `YourTable`

Both of the following statements are equivalent:

```
SELECT * FROM MyTable (id INTEGER)
SELECT * FROM MYTABLE (id INTEGER)
```

The second statement below will fail with `TABLE NOT FOUND` since there is no table named `YOURTABLE`:

```
SELECT * FROM "YourTable" (id INTEGER)   // executes without error
SELECT * FROM YourTable (id INTEGER)     // error, table not found
```

All DB2 CLI catalog function arguments treat the names of objects as *case sensitive*, that is, as if each name was delimited.

SQLDriverConnect() Versus SQLConnect():

Using `SQLDriverConnect()` allows the application to rely on the dialog box provided by DB2 CLI to prompt the user for the connection information.

If an application uses its own dialog boxes to query the connect information, the user should be able to specify additional connect options in the connection string. The string should also be stored and used as a default on subsequent connections.

SQL Governor Implementation:

Each time an SQL statement is prepared, the server *estimates* the cost of the statement. The application can then decide whether to continue with the execution of the statement.

This estimate can be obtained from the SQLCA (SQLERRD(4)), and used by the application directly or the SQL_ATTR_DB2ESTIMATE connection attribute can be set to a threshold value. If the estimated cost of any statement exceeds the threshold, DB2 CLI displays a dialog box with a warning and a prompt to continue or cancel the execution of the statement.

The suggested threshold value is 60000, although in general the application should allow the end user to set the threshold value.

Note: The estimate is only an estimate of the total resources used by the server to execute the statement, it does not indicate the time required to execute the statement.

An estimate of the number of rows in the result is also available from the SQLCA (SQLERRD(3)), and could also be used by the application to restrict large queries.

Note: The accuracy of the information returned in the SQLERRD(3) and SQLERRD(4) fields depends on many factors such as the use of parameter markers and expressions within the statement. If the database statistics in the catalog tables are up to date, they will provide much more accurate information. You can update the database statistics on DB2 Universal Database by issuing the RUNSTATS command from a command line processor session.

Turning Off Statement Scanning:

DB2 CLI by default, scans each SQL statement searching for vendor escape clause sequences.

If the application does not generate SQL statements that contain vendor escape clause sequences, then the SQL_ATTR_NOSCAN statement attribute should be set to SQL_NOSCAN_ON at the connection level so that DB2 CLI does not perform a scan for vendor escape clauses.

Holding Cursors Across Rollbacks:

Applications that need to deal with complex transaction management issues may benefit from establishing multiple concurrent connections to the same database. Each connection in DB2 CLI has its own transaction scope, so any actions performed on one connection do not affect the transactions of other connections.

For example, all open cursors within a transaction get closed if a problem causes the transaction to be rolled back. An application can use multiple connections to the same database to separate statements with open cursors; since the cursors are in separate transactions, a rollback on one statement does not affect the cursors of the other statements.

However, using multiple connections may mean bringing some data across to the client on one connection, and then sending it back to the server on the other connection. For example:

- Suppose in connection #1 you are accessing large object columns and have created LOB locators that map to portions of large object values.
- If in connection #2, you wish to use (e.g. insert) the portion of the LOB values represented by the LOB locators, you would have to move the LOB values in connection #1 first to the application, and then pass them to the tables that you are working with in connection #2. This is because connection #2 does not know anything about the LOB locators in connection #1.
- If you only had one connection, then you could just use the LOB locators directly. However, you would lose the LOB locators as soon as you rolled back your transaction.

Note: When multiple connections to a single database are used by an application, the application must be careful to synchronize access to database objects or it may experience various lock contention issues, as database locks are not shared between transactions. Updates by one connection can easily force other connections into a lock-wait state until the first connection releases the lock (through a COMMIT or ROLLBACK).

Preparing Compound SQL Sub-statements:

In order to maximize efficiency of the compound statement, sub-statements should be prepared before the BEGIN COMPOUND statement, and then executed within the compound statement.

This also simplifies error handling since prepare errors can be handled outside of the compound statement.

User Defined Types and Casting:

If a parameter marker is used in a predicate of a query statement, and the parameter is a user defined type, the statement must use a CAST function to cast either the parameter marker or the UDT.

For example, suppose the following type and table is defined:

```
CREATE DISTINCT TYPE CNUM AS INTEGER WITH COMPARISONS

CREATE TABLE CUSTOMER (
        Cust_Num     CNUM NOT NULL,
        First_Name   CHAR(30) NOT NULL,
        Last_Name    CHAR(30) NOT NULL,
        Phone_Num    CHAR(20) WITH DEFAULT,
        PRIMARY KEY  (Cust_Num) )
```

Suppose also that the following SQL statement was then issued:

```
SELECT first_name, last_name, phone_num from customer
WHERE cust_num = ?
```

This statement would fail because the parameter marker cannot be of type CNUM and thus the comparison fails due to incompatible types.

Casting the column to integer (its base SQL type), allows the comparison to work since a parameter can be provided for type integer:

```
SELECT  first_name, last_name, phone_num from customer
where cast( cust_num as integer ) = ?
```

Alternatively the parameter marker can be cast to INTEGER and the server can then apply the INTEGER to CNUM conversion:

```
SELECT  first_name, last_name, phone_num FROM customer
where cust_num = cast( ? as integer )
```

Deferred Prepare to Reduce Network Flow:

In DB2 CLI, deferred prepare is on by default. The PREPARE request is not sent to the server until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance. This is of greatest benefit when the application generates queries where the answer set is very small, and the overhead of separate requests and replies is not spread across multiple blocks of query data. In an environment where a DB2 Connect or DDCS gateway is used, there is a greater opportunity for cost reduction because four request and reply combinations are reduced to two.

Note: Functions such as `SQLDescribeParam()`, `SQLDescribeCol()`, `SQLNumParams()`, and `SQLNumResultCols()` require that the statement has been prepared. If the statement has not already been prepared, these functions trigger an immediate PREPARE request to the server, and the benefit of deferred prepare does not appear.

Related concepts:

- "Catalog Functions for Querying System Catalog Information in CLI Applications" on page 195
- "Vendor Escape Clauses in CLI Applications" on page 199
- "Handles in CLI" on page 19
- "Cursors in CLI Applications" on page 75
- "Large Object Usage in CLI Applications" on page 116

Related tasks:

- "Executing SQL Statements in CLI Applications with Compound SQL" on page 147
- "Retrieving Array Data in CLI Applications Using Column-wise Binding" on page 104
- "Retrieving Array Data in CLI Applications Using Row-wise Binding" on page 106

Related reference:

- "CLI and ODBC Function Summary" in the *CLI Guide and Reference, Volume 2*
- "SQLConnect Function (CLI) - Connect to a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLDriverConnect Function (CLI) - (Expanded) Connect to a Data Source" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Chapter 5. Cursors

Cursors

Cursors in CLI Applications

A CLI application uses a cursor to retrieve rows from a result set. A cursor is a moveable pointer to a row in the result table of an active query statement. The UDB client for the Unix and Windows® platforms supports updatable scrollable cursors when run against Version 8 of DB2® for Unix and Windows and DB2 for z/OS™. To access a scrollable cursor in a three-tier environment on DB2 for z/OS or DB2 for OS/390 Version 7 and above, the client and the gateway must be running DB2 UDB Version 7.2 or above.

A cursor is opened when a dynamic SQL SELECT statement is successfully executed by `SQLExecute()` or `SQLExecDirect()`. There is typically a one-to-one correlation between application cursor operations and the operations performed by the DB2 CLI driver with the cursor. Immediately after the successful execution, the cursor is positioned before the first row of the result set, and FETCH operations through calls to `SQLFetch()`, `SQLFetchScroll()`, or `SQLExtendedFetch()` will advance the cursor one row at a time through the result set. When the cursor has reached the end of the result set, the next fetch operation will return SQLCODE +100. From the perspective of the CLI application, `SQLFetch()` returns SQL_NO_DATA_FOUND when the end of the result set is reached.

Types of Cursors:

There are two types of cursors supported by DB2 CLI:

nonscrollable
Forward-only nonscrollable cursors are the default cursor type used by the DB2 CLI driver. This cursor type is unidirectional and requires the least amount of overhead processing.

scrollable

There are two types of scrollable cursors supported by DB2 CLI:

static This is a read-only cursor. Once it is created, no rows can be added or removed, and no values in any rows will change. The cursor is not affected by other applications accessing the same data. The isolation level of the statement used to create the cursor determines how the rows of the cursor are locked, if at all.

keyset-driven

Unlike a static scrollable cursor, a keyset-driven scrollable cursor can detect and make changes to the underlying data. Keyset cursors are based on row keys. When a keyset-driven cursor is first opened, it stores the keys in a keyset for the life of the entire result set. The keyset is used to determine the order and set of rows that are included in the cursor. As the cursor scrolls through the result set, it uses the keys in this keyset to retrieve the most recent values in the database, which are not necessarily the values that existed when the cursor was first opened. For this reason, changes are not reflected until the application scrolls to the row.

There are various types of changes to the underlying data that a keyset-driven cursor may or may not reflect:

- Changed values in existing rows. The cursor will reflect these types of changes. Because the cursor fetches a row from the database each time it is required, keyset-driven cursors always detect changes made by themselves and other cursors.
- Deleted rows. The cursor will reflect these types of changes. If a selected row in the rowset is deleted after the keyset is generated, it will appear as a "hole" in the cursor. When the cursor goes to fetch the row again from the database, it will realize that the row is no longer there.
- Added rows. The cursor will not reflect these types of changes. The set of rows is determined once, when the cursor is first opened. To see the inserted rows, the application must re-execute the query.

Note: DB2 CLI currently only supports keyset-driven cursors if the server supports them. The DB2 Version 8 server now supports updatable scrollable cursors, so applications that require keyset cursor functionality and currently access DB2 for OS/390 Version 6 or DB2 for Unix and Windows Version 7 and earlier should not migrate their clients to DB2 Version 8.

Cursor Attributes:

The table below lists the default attributes for cursors in DB2 CLI.

Table 8. Default attributes for cursors in CLI

Cursor Type	Cursor Sensitivity	Cursor Updatable	Cursor Concurrency	Cursor Scrollable
forward-only[a]	unspecified	non-updatable	read-only concurrency	non-scrollable
static	insensitive	non-updatable	read-only concurrency	scrollable
keyset-driven	sensitive	updatable	values concurrency	scrollable

a Forward-only is the default behavior for a scrollable cursor without the `FOR UPDATE` clause. Specifying `FOR UPDATE` on a forward-only cursor creates an updatable, lock concurrency, non-scrollable cursor.

Update of Keyset-Driven Cursors:

A keyset-driven cursor is an updatable cursor. The CLI driver appends the `FOR UPDATE` clause to the query, except when the query is issued as a `SELECT ... FOR READ ONLY` query, or if the `FOR UPDATE` clause already exists. The keyset-driven cursor is a values concurrency cursor. A values concurrency cursor results in optimistic locking, where locks are not held until an update or delete is attempted. When an update or delete is attempted, the database server compares the previous values the application retrieved to the current values in the underlying table. If the values match, then the update or delete succeeds. If the values do not match, then the operation fails. If failure occurs, the application should query the values again and re-issue the update or delete if it is still applicable.

An application can update a keyset-driven cursor in two ways:

- Issue an `UPDATE WHERE CURRENT OF <cursor name>` or `DELETE WHERE CURRENT OF <cursor name>` using `SQLPrepare()` with `SQLExecute()` or `SQLExecDirect()`
- Use `SQLSetPos()` or `SQLBulkOperations()` to update, delete, or add a row to the result set.

  Note: Rows added to a result set via `SQLSetPos()` or `SQLBulkOperations()` are inserted into the table on the server, but are not added to the server's result set. Therefore, these rows are not updatable nor are they sensitive to changes made by other transactions. The inserted rows will appear, however, to be part of the result set, since they are cached on the client. Any triggers that apply to the inserted rows will appear to the application as if they have not been applied. To make the inserted rows updatable, sensitive, and to see the result of applicable triggers, the application must issue the query again to regenerate the result set.

Related concepts:
- "Cursor Considerations for CLI Applications" on page 78
- "Result Set Terminology in CLI Applications" on page 80

Related reference:
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLExtendedFetch Function (CLI) - Extended Fetch (Fetch Array of Rows)" in the *CLI Guide and Reference, Volume 2*
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "DELETE statement" in the *SQL Reference, Volume 2*
- "UPDATE statement" in the *SQL Reference, Volume 2*
- "CURSORHOLD CLI/ODBC Configuration Keyword" on page 309

Cursor Considerations for CLI Applications

Which Cursor Type to Use:

The first decision to make is between a forward-only cursor and a scrollable cursor. A forward-only cursor incurs less overhead than a scrollable cursor, and scrollable cursors have the potential for decreased concurrency. If your application does not need the additional features of a scrollable cursor, then you should use a nonscrollable cursor.

If a scrollable cursor is required then you have to decide between a static cursor or a keyset-driven cursor. A static cursor involves the least overhead. If the application does not need the additional features of a keyset-driven cursor then a static cursor should be used.

If the application needs to detect changes to the underlying data or needs to add, update, or delete data from the cursor, then the application must use a keyset-driven cursor.

To determine the attributes of the types of cursors supported by the driver and DBMS, the application should call `SQLGetInfo()` with an *InfoType* of:

- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2

Unit of Work Considerations:

A cursor can be closed either explicitly or implicitly. An application can explicitly close a cursor by calling `SQLCloseCursor()`. Any further attempts to manipulate the cursor will result in error, unless the cursor is opened again. The implicit closure of a cursor depends on a several factors including how the cursor was declared and whether or not a COMMIT or ROLLBACK occurs.

By default, the DB2 CLI driver declares all cursors as WITH HOLD. This means that any open cursor will persist across COMMITs, thereby requiring the application to explicitly close each cursor. Be aware, however, that if a cursor is closed in autocommit mode, then any other open cursors that are not defined with the WITH HOLD option will be closed and all remaining open cursors will become unpositioned. (This means that no positioned updates or deletes can be performed without issuing another fetch.) There are two ways to change whether a cursor is declared WITH HOLD:

- Set the statement attribute SQL_ATTR_CURSOR_HOLD to SQL_CURSOR_HOLD_ON (default) or SQL_CURSOR_HOLD_OFF. This setting only affects cursors opened on the statement handle after this value has been set. It will not affect cursors already open.
- Set the CLI/ODBC configuration keyword CURSORHOLD to change the default DB2 CLI driver behavior. Setting CURSORHOLD=1 preserves the default behavior of cursors declared as WITH HOLD, and CURSORHOLD=0 results in cursors being closed when each transaction is committed. You can override this keyword by setting the SQL_ATTR_CURSOR_HOLD statement attribute described above.

Note: A ROLLBACK will close all cursors, including those declared WITH HOLD.

Troubleshooting for Applications Created Before Scrollable Cursor Support:

Because scrollable cursor support is a newer feature, some CLI/ODBC applications that were working with previous releases of DB2® for OS/390 or DB2 for Unix and Windows® may encounter behavioral or performance changes. This occurs because before scrollable cursors were supported, applications that requested a scrollable cursor would receive a forward-only cursor. To restore an application's previous behavior before scrollable cursor support, set the following configuration keywords in the `db2cli.ini` file:

Table 9. Configuration keyword values restoring application behavior before scrollable cursor support

Configuration Keyword Setting	**Description**
`PATCH2=6`	Returns a message that scrollable cursors (both keyset-driven and static) are not supported. CLI automatically downgrades any request for a scrollable cursor to a forward-only cursor.
`DisableKeysetCursor=1`	Disables keyset-driven scrollable cursors. This can be used to force the CLI driver to give the application a static cursor when a keyset-driven cursor is requested.

Related concepts:

- "Commit Modes in CLI Applications" on page 38
- "Cursors in CLI Applications" on page 75
- "Result Set Terminology in CLI Applications" on page 80

Related reference:

- "SQLGetInfo Function (CLI) - Get General Information" in the *CLI Guide and Reference, Volume 2*
- "COMMIT statement" in the *SQL Reference, Volume 2*
- "ROLLBACK statement" in the *SQL Reference, Volume 2*
- "CURSORHOLD CLI/ODBC Configuration Keyword" on page 309
- "DISABLEKEYSETCURSOR CLI/ODBC Configuration Keyword" on page 317
- "PATCH2 CLI/ODBC Configuration Keyword" on page 326

Result Sets

Result Set Terminology in CLI Applications

The following terms describe result handling:

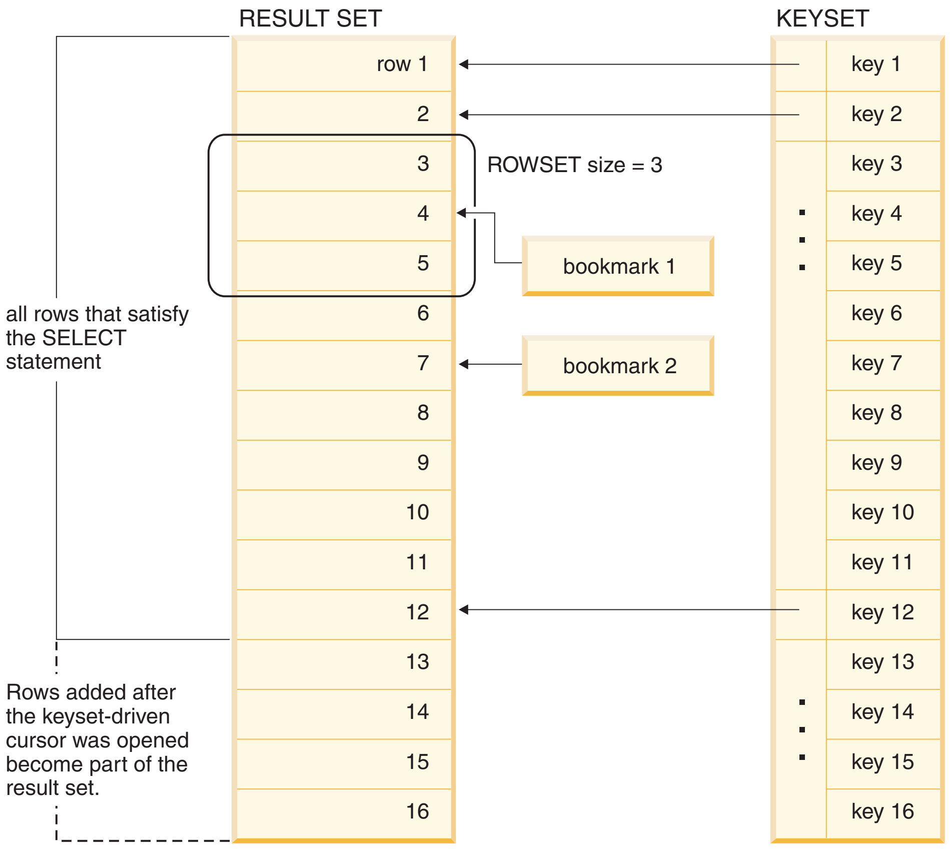
result set
The complete set of rows that satisfy the SQL SELECT statement. This is the set from which fetches retrieve rows to populate the rowset.

rowset
The subset of rows from the result set that is returned after each fetch. The application indicates the size of the rowset before the first fetch of data, and can modify the size before each subsequent fetch. Each call to `SQLFetch()`, `SQLFetchScroll()`, or `SQLExtendedFetch()` populates the rowset with the appropriate rows from the result set.

bookmark
It is possible to store a reference to a specific row in the result set called a bookmark. Once stored, the application can continue to move through the result set, then return to the bookmarked row to generate a rowset. You can also use a bookmark to perform updates and deletions with `SQLBulkOperations()`.

keyset A set of key values used to identify the set and order of rows that are included in a keyset-driven cursor. The keyset is created when a keyset-driven cursor is first opened. As the cursor scrolls through the result set, it uses the keys in the keyset to retrieve the current data values for each row.

The following figure demonstrates the relationship between the terms described above:



Related concepts:

- "Cursors in CLI Applications" on page 75
- "Cursor Considerations for CLI Applications" on page 78
- "Bookmarks in CLI Applications" on page 90

Related reference:

- "SQLExtendedFetch Function (CLI) - Extended Fetch (Fetch Array of Rows)" in the *CLI Guide and Reference, Volume 2*
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SELECT statement" in the *SQL Reference, Volume 2*

- “SQLBulkOperations Function (CLI) - Add, Update, Delete or Fetch a Set of Rows” in the *CLI Guide and Reference, Volume 2*

Rowset Retrieval Examples in CLI Applications

Partial Rowset Example:

When working with rowsets, you should verify what portion of the result set returned contains meaningful data. The application cannot assume that the entire rowset will contain data. It must check the row status array after each rowset is created to determine the number of rows returned, because there are instances where the rowset will not contain a complete set of rows. For instance, consider the case where the rowset size is set to 10, and `SQLFetchScroll()` is called using SQL_FETCH_ABSOLUTE and *FetchOffset* is set to -3. This will attempt to return 10 rows starting 3 rows from the end of the result set. Only the first three rows of the rowset will contain meaningful data, however, and the application must ignore the rest of the rows.

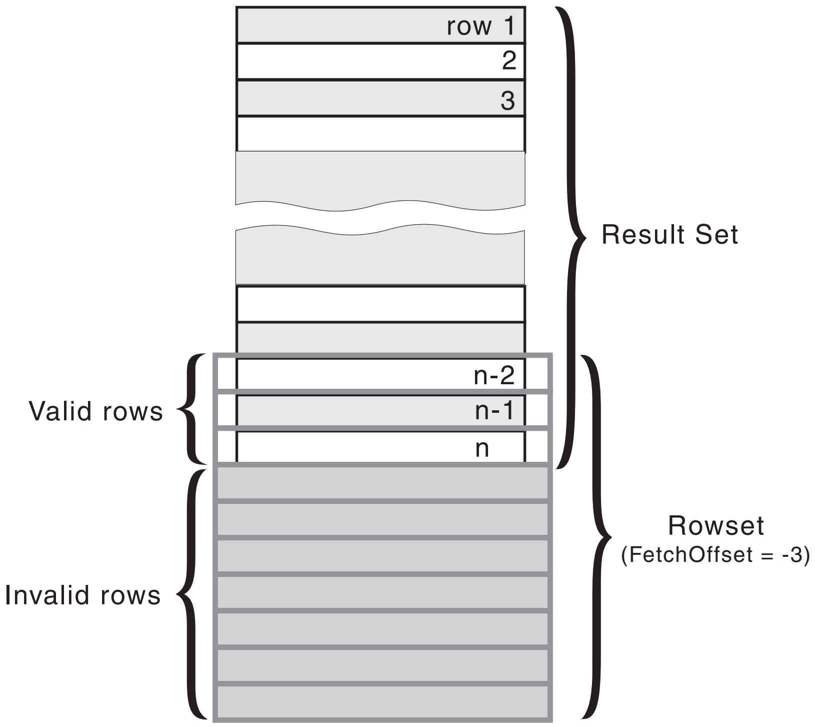


Figure 4. Partial Rowset Example

Fetch Orientations Example:

The following figure demonstrates a number of calls to `SQLFetchScroll()` using various *FetchOrientation* values. The result set includes all of the rows (from 1 to n), and the rowset size is 3. The order of the calls is indicated on the left, and the *FetchOrientation* values are indicated on the right.

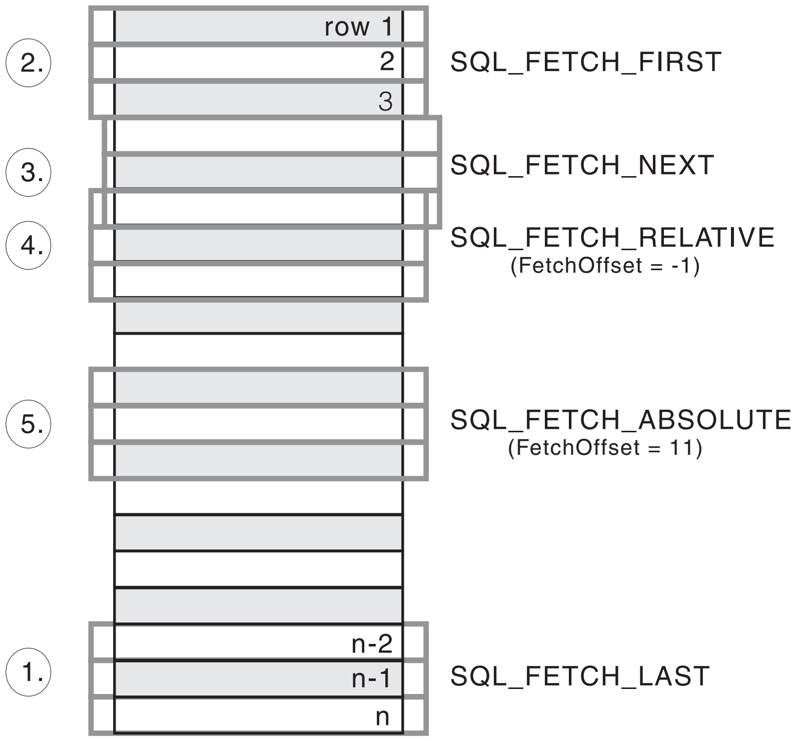


Figure 5. Example of Retrieving Rowsets

Related concepts:

- "Result Set Terminology in CLI Applications" on page 80

Related reference:

- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*

Specifying the Rowset Returned from the Result Set

Before you begin to retrieve data, you need to establish the rowset that will be returned. This topic describes the steps associated with setting up the rowset.

Prerequisites:

Before specifying the rowset, ensure that you have initialized your CLI application.

Procedure:

DB2 CLI allows an application to specify a rowset for a non-scrollable or scrollable cursor that spans more than one row at a time. To effectively work with a rowset, an application should perform the following:

1. Specify the size of the rowset returned from calls to `SQLFetch()` or `SQLFetchScroll()` by setting the statement attribute SQL_ATTR_ROW_ARRAY_SIZE to the number of rows in the rowset. The default number of rows is 1. For example, to delcare a rowset size of 35 rows, issue the following call:
   ```
   #define ROWSET_SIZE 35
   /* ... */
   rc = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROW_ARRAY_SIZE,
                       (SQLPOINTER) ROWSET_SIZE,
                       0);
   ```
2. Set up a variable that will store the number of rows returned. Declare a variable of type SQLUINTEGER and set the SQL_ATTR_ROWS_FETCHED_PTR statement attribute to point to this variable. In the following example, *rowsFetchedNb* will hold the number of rows returned in the rowset after each call to `SQLFetchScroll()`:
   ```
   /* ... */

   SQLUINTEGER rowsFetchedNb;

   /* ... */

   rc = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROWS_FETCHED_PTR,
                       &rowsFetchedNb,
                       0);
   ```
3. Set up the row status array. Declare an array of type SQLUSMALLINT with the same number of rows as the size of the rowset (as determined in Step 1). Then specify the address of this array with the statement attribute SQL_ATTR_ROW_STATUS_PTR. For example:
   ```
   /* ... */
   SQLUSMALLINT    row_status[ROWSET_SIZE];
   /* ... */
   /* Set a pointer to the array to use for the row status */
   rc = SQLSetStmtAttr(
                hstmt,
                SQL_ATTR_ROW_STATUS_PTR,
                (SQLPOINTER) row_status,
                0);
   ```

The row status array provides additional information about each row in the rowset. After each call to `SQLFetch()` or `SQLFetchScroll()`, the array is updated. If the call to `SQLFetch()` or `SQLFetchScroll()` does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, then the contents of the row status array are undefined. Otherwise, any of the row status array values will be returned (refer to the row status array section of the SQLFetchScroll() documentation for a complete list of values).

4. Position the rowset within the result set, indicating the position you want the rowset to begin. Specify this position by calling `SQLFetch()`, or `SQLFetchScroll()` with *FetchOrientation* and *FetchOffset* values. For example, the following call generates a rowset starting on the 11th row in the result set:

```
SQLFetchScroll(hstmt,   /* Statement handle */
    SQL_FETCH_ABSOLUTE, /* FetchOrientation value */
    11);                /* Offset value */
```

Scroll bar operations of a screen-based application can be mapped directly to the positioning of a rowset. By setting the rowset size to the number of lines displayed on the screen, the application can map the movement of the scroll bar to calls to `SQLFetchScroll()`.

Note: If the application can buffer data in the display and regenerate the result set to see updates, then use a forward-only cursor instead. This yields better performance for small result sets.

Rowset Retrieved	FetchOrientation Value	Scroll bar
First rowset	SQL_FETCH_FIRST	Home: Scroll bar at the top
Last rowset	SQL_FETCH_LAST	End: Scroll bar at the bottom
Next rowset	SQL_FETCH_NEXT (same as calling `SQLFetch()`)	Page Down
Previous rowset	SQL_FETCH_PRIOR	Page Up
Rowset starting on next row	SQL_FETCH_RELATIVE with *FetchOffset* set to 1	Line Down
Rowset starting on previous row	SQL_FETCH_RELATIVE with *FetchOffset* set to -1	Line Up
Rowset starting on a specific row	SQL_FETCH_ABSOLUTE with *FetchOffset* set to an offset from the start (a positive value) or the end (a negative value) of the result set	Application generated
Rowset starting on a previously bookmarked row	SQL_FETCH_BOOKMARK with *FetchOffset* set to a positive or negative offset from the bookmarked row	Application generated

5. Check the rows fetched pointer after each rowset is created to determine the number of rows returned. Check the row status array for the status of each row, because there are instances where the rowset will not contain a complete set of rows. The application cannot assume that the entire rowset will contain data.

   For instance, consider the case where the rowset size is set to 10, and `SQLFetchScroll()` is called using SQL_FETCH_ABSOLUTE and *FetchOffset* is set to -3. This will attempt to return 10 rows starting 3 rows from the end of the result set. Only the first three rows of the rowset will contain meaningful data, however, and the application must ignore the rest of the rows.

Related concepts:

- "Cursor Considerations for CLI Applications" on page 78
- "Result Set Terminology in CLI Applications" on page 80

Related tasks:

- "Initializing CLI Applications" on page 24
- "Retrieving Data with Scrollable Cursors in a CLI Application" on page 87
- "Retrieving Data with Bookmarks in a CLI Application" on page 91

Related reference:

- "C Data Types for CLI Applications" on page 54
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Related samples:

- "tbread.c -- How to read data from tables (CLI)"

Retrieving Data with Scrollable Cursors in a CLI Application

Scrollable cursors allow you to move throughout a result set. You can make use of this feature when retrieving data. This topic describes how to use scrollable cursors to retrieve data.

Prerequisites:

Before you retrieve data using scrollable cursors, ensure that you have initialized your CLI application.

Procedure:

To use scrollable cursors to retrieve data:

1. Specify the size of the rowset returned by setting the statement attribute SQL_ATTR_ROW_ARRAY_SIZE to the number of rows in the rowset. The default number of rows is 1. For example, to declare a rowset size of 35 rows, issue the following call:
   ```
   #define ROWSET_SIZE 35
   /* ... */
   rc = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROW_ARRAY_SIZE,
                       (SQLPOINTER) ROWSET_SIZE,
                       0);
   ```
2. Specify the type of scrollable cursor to use. Using `SQLSetStmtAttr()`, set the SQL_ATTR_CURSOR_TYPE statement attribute to SQL_CURSOR_STATIC for a static read-only cursor or to SQL_CURSOR_KEYSET_DRIVEN for a keyset-driven cursor. For example:
   ```
   sqlrc = SQLSetStmtAttr (hstmt,
                           SQL_ATTR_CURSOR_TYPE,
                           (SQLPOINTER) SQL_CURSOR_STATIC,
                           0);
   ```

   If the type of cursor is not set, the default forward-only non-scrollable cursor will be used.
3. Set up a variable that will store the number of rows returned. Declare a variable of type SQLUINTEGER and set the SQL_ATTR_ROWS_FETCHED_PTR statement attribute to point to this variable. In the following example, *rowsFetchedNb* will hold the number of rows returned in the rowset after each call to `SQLFetchScroll()`:
   ```
   /* ... */

   SQLUINTEGER rowsFetchedNb;

   /* ... */

   rc = SQLSetStmtAttr(hstmt,
                       SQL_ATTR_ROWS_FETCHED_PTR,
                       &rowsFetchedNb,
                       0);
   ```
4. Set up the row status array. Declare an array of type SQLUSMALLINT with the same number of rows as the size of the rowset (as determined in Step 1). Then specify the address of this array with the statement attribute SQL_ATTR_ROW_STATUS_PTR. For example:

```
/* ... */
SQLUSMALLINT    row_status[ROWSET_SIZE];
/* ... */
/* Set a pointer to the array to use for the row status */
rc = SQLSetStmtAttr(
            hstmt,
            SQL_ATTR_ROW_STATUS_PTR,
            (SQLPOINTER) row_status,
            0);
```

The row status array provides additional information about each row in the rowset. After each call to `SQLFetchScroll()`, the array is updated. If the call to `SQLFetchScroll()` does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, then the contents of the row status array are undefined. Otherwise, any of the row status array values will be returned (refer to the row status array section of the SQLFetchScroll() documentation for a complete list of values).

5. Optional: If you want to use bookmarks with the scrollable cursor, set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE. For example:

```
sqlrc = SQLSetStmtAttr (hstmt,
                        SQL_ATTR_USE_BOOKMARKS,
                        (SQLPOINTER) SQL_UB_VARIABLE,
                        0);
```

6. Issue an SQL SELECT statement.
7. Execute the SQL SELECT statement.
8. Bind the result set using either column-wise or row-wise binding.
9. Fetch a rowset of rows from the result set.
   a. Call `SQLFetchScroll()` to fetch a rowset of data from the result set. Position the rowset within the result set indicating the position you want the rowset to begin. Specify this position by calling `SQLFetchScroll()` with *FetchOrientation* and *FetchOffset* values. For example, the following call generates a rowset starting on the 11th row in the result set:

```
SQLFetchScroll(hstmt,              /* Statement handle */
               SQL_FETCH_ABSOLUTE, /* FetchOrientation value */
               11);                /* Offset value */
```

   b. Check the row status array after each rowset is created to determine the number of rows returned, because there are instances where the rowset will not contain a complete set of rows. The application cannot assume that the entire rowset will contain data.

      For instance, consider the case where the rowset size is set to 10, and `SQLFetchScroll()` is called using SQL_FETCH_ABSOLUTE and *FetchOffset* is set to -3. This will attempt to return 10 rows starting 3 rows from the end of the result set. Only the first three rows of the rowset will contain meaningful data, however, and the application must ignore the rest of the rows.

   c. Display or manipulate the data in the rows returned.

10. Close the cursor by calling `SQLCloseCursor()` or free the statement handle by calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_STMT.

    Freeing the statement handles is not required every time retrieval has finished. The statement handles can be freed at a later time, when the application is freeing other handles.

Related concepts:
- "Cursors in CLI Applications" on page 75
- "Cursor Considerations for CLI Applications" on page 78
- "Bookmarks in CLI Applications" on page 90

Related tasks:
- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Issuing SQL Statements in CLI Applications" on page 29

Related reference:
- "C Data Types for CLI Applications" on page 54
- "SQLCloseCursor Function (CLI) - Close Cursor and Discard Pending Results" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Related samples:
- "tbread.c -- How to read data from tables (CLI)"

Bookmarks

Bookmarks in CLI Applications

When scrollable cursors are used, you can save a reference to any row in the result set using a bookmark. The application can then use that bookmark as a relative position to retrieve a rowset of information, or to update or delete a row when using keyset cursors. You can retrieve a rowset starting from the bookmarked row, or specify a positive or negative offset.

Once you have positioned the cursor to a row in a rowset using `SQLSetPos()`, you can obtain the bookmark value starting from column 0 using `SQLGetData()`. In most cases you will not want to bind column 0 and retrieve the bookmark value for every row, but use `SQLGetData()` to retrieve the bookmark value for the specific row you require.

A bookmark is only valid within the result set in which it was created. The bookmark value will be different if you select the same row from the same result set in two different cursors.

The only valid comparison is a byte-by-byte comparison between two bookmark values obtained from the same result set. If they are the same then they both point to the same row. Any other mathematical calculations or comparisons between bookmarks will not provide any useful information. This includes comparing bookmark values within a result set, and between result sets.

Related concepts:

- "Cursor Considerations for CLI Applications" on page 78
- "Result Set Terminology in CLI Applications" on page 80

Related reference:

- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLSetPos Function (CLI) - Set the Cursor Position in a Rowset" in the *CLI Guide and Reference, Volume 2*

Retrieving Data with Bookmarks in a CLI Application

Bookmarks, available only when scrollable cursors are used, allow you to save a reference to any row in a result set. You can take advantage of this feature when retrieving data. This topic describes how to retrieve data using bookmarks.

Prerequisites:

Before you retrieve data with bookmarks, ensure that you have initialized your CLI application. The steps explained here should be performed in addition to those described in ″Retrieving Data with Scrollable Cursors in a CLI Application″.

Procedure:

To use bookmarks with scrollable cursors to retrieve data:

1. Indicate that bookmarks will be used (if not already done so) by setting the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE. For example:

   ```
   sqlrc = SQLSetStmtAttr (hstmt,
                           SQL_ATTR_USE_BOOKMARKS,
                           (SQLPOINTER) SQL_UB_VARIABLE,
                           0);
   ```

2. Get the bookmark value from the desired row in the rowset after executing the SELECT statement and retrieving the rowset using `SQLFetchScroll()`. Do this by calling `SQLSetPos()` to position the cursor within the rowset. Then call `SQLGetData()` to retrieve the bookmark value. For example:

   ```
     sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_ABSOLUTE, 15);
     /* ... */
     sqlrc = SQLSetPos(hstmt, 3, SQL_POSITION, SQL_LOCK_NO_CHANGE);
     /* ... */
     sqlrc = SQLGetData(hstmt, 0, SQL_C_LONG, bookmark.val, 4,
                        &bookmark.ind);
   ```

   In most cases, you will not want to bind column 0 and retrieve the bookmark value for every row, but use `SQLGetData()` to retrieve the bookmark value for the specific row you require.

3. Store the bookmark location for the next call to `SQLFetchScroll()`. Set the SQL_ATTR_FETCH_BOOKMARK statement attribute to the variable that contains the bookmark value. For example, continuing from the example above, *bookmark.val* stores the bookmark value, so call `SQLSetStmtAttr()` as follows:

   ```
     sqlrc = SQLSetStmtAttr(hstmt,
                            SQL_ATTR_FETCH_BOOKMARK_PTR,
                            (SQLPOINTER) bookmark.val,
                            0);
   ```

4. Retrieve a rowset based on the bookmark. Once the bookmark value is stored, the application can continue to use `SQLFetchScroll()` to retrieve data from the result set. The application can then move throughout the result set, but still retrieve a rowset based on the location of the bookmarked row at any point before the cursor is closed.

   The following call to `SQLFetchScroll()` retrieves a rowset starting from the bookmarked row:

   ```
     sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_BOOKMARK, 0);
   ```

   The value 0 specifies the offset. You would specify -3 to begin the rowset 3 rows before the bookmarked row, or specify 4 to begin 4 rows after. For example, the following call from retrieves a rowset 4 rows after the bookmarked row:

   ```
     sqlrc = SQLFetchScroll(hstmt, SQL_FETCH_BOOKMARK, 4);
   ```

Note that the variable used to store the bookmark value is not specified in the `SQLFetchScroll()` call. It was set in the previous step using the statement attribute SQL_ATTR_FETCH_BOOKMARK_PTR.

Related concepts:

- "Cursor Considerations for CLI Applications" on page 78
- "Result Set Terminology in CLI Applications" on page 80
- "Bookmarks in CLI Applications" on page 90

Related tasks:

- "Initializing CLI Applications" on page 24
- "Retrieving Data with Scrollable Cursors in a CLI Application" on page 87

Related reference:

- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLSetPos Function (CLI) - Set the Cursor Position in a Rowset" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "tbread.c -- How to read data from tables (CLI)"

Chapter 6. Array Input and Output

Array Input

Binding Parameter Markers in CLI Applications with Column-Wise Array Input

To process an SQL statement that will be repeated with different values, you can use column-wise array input to achieve bulk inserts, deletes, or updates. This results in fewer network flows to the server because `SQLExecute()` does not have to be called repeatedly on the same SQL statement for each value. Column-wise array input allows arrays of storage locations to be bound to parameter markers. A different array is bound to each parameter.

Prerequisites:

Before binding parameter markers with column-wise binding, ensure that you have initialized your CLI application.

Restrictions:

For character and binary input data, the application uses the maximum input buffer size argument (*BufferLength*) of the `SQLBindParameter()` call to indicate to DB2 CLI the location of values in the input array. For other input data types, the length of each element in the array is assumed to be the size of the C data type.

Procedure:

To bind parameter markers using column-wise array input:

1. Specify the size of the arrays (the number rows to be inserted) by calling `SQLSetStmtAttr()` with the SQL_ATTR_PARAMSET_SIZE statement attribute.

2. Initialize and populate an array for each parameter marker to be bound.

   Note: Each array must contain at least SQL_ATTR_PARAMSET_SIZE elements, otherwise, memory access violations may occur.
3. Optional: Indicate that column-wise binding is to be used by setting the SQL_ATTR_BIND_TYPE statement attribute to SQL_PARAMETER_BIND_BY_COLUMN (this is the default setting).
4. Bind each parameter marker to its corresponding array of input values by calling `SQLBindParameter()` for each parameter marker.

Related concepts:
- "Parameter Marker Binding in CLI Applications" on page 33
- "Parameter Diagnostic Information in CLI Applications" on page 98

Related tasks:
- "Initializing CLI Applications" on page 24

Related reference:
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Binding Parameter Markers in CLI Applications with Row-Wise Array Input

To process an SQL statement that will be repeated with different values, you can use row-wise array input to achieve bulk inserts, deletes, or updates. This results in fewer network flows to the server because `SQLExecute()` does not have to be called repeatedly on the same SQL statement for each value. Row-wise array input allows an array of structures to be bound to parameters.

Prerequisites:

Before binding parameter markers with row-wise binding, ensure that you have initialized your CLI application.

Procedure:

To bind parameter markers using row-wise array input:

1. Initialize and populate an array of structures that contains two elements for each parameter: the first element contains the length/indicator buffer, and the second element holds the value itself. The size of the array corresponds to the number of values to be applied to each parameter. For example, the following array contains the length and value for three parameters:
   ```
   struct { SQLINTEGER La; SQLINTEGER A;  /* Information for parameter A */
            SQLINTEGER Lb; SQLCHAR B[4];  /* Information for parameter B */
            SQLINTEGER Lc; SQLCHAR C[11]; /* Information for parameter C */
          } R[n];
   ```
2. Indicate that row-wise binding is to by used by setting the SQL_ATTR_PARAM_BIND_TYPE statement attribute to the length of the struct created in the previous step, using `SQLSetStmtAttr()`.
3. Set the statement attribute SQL_ATTR_PARAMSET_SIZE to the number of rows of the array, using `SQLSetStmtAttr()`.
4. Bind each parameter to the first row of the array created in step 1 using `SQLBindParameter()`. For example,
   ```
   /* Parameter A */
   rc = SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
            SQL_INTEGER, 5, 0, &R[0].A, 0, &R.La);

   /* Parameter B */
   rc = SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
            10, 0, R[0].B, 10, &R.Lb);

   /* Parameter C */
   rc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
            3, 0, R[0].C, 3, &R.Lc);
   ```

Related concepts:

- "Parameter Marker Binding in CLI Applications" on page 33
- "Parameter Diagnostic Information in CLI Applications" on page 98

Related tasks:

- "Initializing CLI Applications" on page 24

Related reference:

- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Parameter Diagnostic Information in CLI Applications

A *parameter status array* is an array of one or more SQLSMALLINTs allocated by a CLI application. Each element in the array corresponds to an element in the input (or output) parameter array. If specified, the DB2 CLI driver updates the parameter status array with information about the processing status of each set of parameters included in an `SQLExecute()` or `SQLExecDirect()` call.

DB2 CLI updates the elements in the parameter status array with the following values:

- SQL_PARAM_SUCCESS: The SQL statement was successfully executed for this set of parameters.
- SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement was successfully executed for this set of parameters, however, warning information is available in the diagnostics data structure.
- SQL_PARAM_ERROR: An error occurred in processing this set of parameters. Additional error information is available in the diagnostics data structure.
- SQL_PARAM_UNUSED: This parameter set was unused, possibly because a previous parameter set caused an error that aborted further processing.
- SQL_PARAM_DIAG_UNAVAILABLE: Diagnostic information is not available, possibly because an error was detected before the parameter set was even used (for example, an SQL statement syntax error).

A CLI application must call the `SQLSetStmtAttr()` function to set the SQL_ATTR_PARAM_STATUS_PTR attribute before DB2 CLI will update the parameter status array. Alternatively, the application can call the `SQLSetDescField()` function to set the SQL_DESC_ARRAY_STATUS_PTR field in the IPD descriptor to point to the parameter status array.

The statement attribute SQL_ATTR_PARAMS_PROCESSED, or the corresponding IPD descriptor header field SQL_DESC_ROWS_PROCESSED_PTR, can be used to return the number of sets of parameters that have been processed.

Once the application has determined what parameters had errors, it can use the statement attribute SQL_ATTR_PARAM_OPERATION_PTR, or the corresponding APD descriptor header field SQL_DESC_ARRAY_STATUS_PTR, (both of which point to an array of values) to control which sets of parameters are ignored in a second call to `SQLExecute()` or `SQLExecDirect()`.

Related tasks:

- "Binding Parameter Markers in CLI Applications" on page 36

Related reference:

- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLSetDescField Function (CLI) - Set a Single Field of a Descriptor Record" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "Descriptor FieldIdentifier Argument Values (CLI)" in the *CLI Guide and Reference, Volume 2*

Changing Parameter Bindings in CLI Applications with Offsets

When an application needs to change parameter bindings it can call `SQLBindParameter()` a second time. This will change the bound parameter buffer address and the corresponding length/indicator buffer address used. Instead of multiple calls to `SQLBindParameter()`, however, DB2 CLI also supports parameter binding offsets. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLExecute()` or `SQLExecDirect()`.

Prerequisites:

Before changing your parameter bindings, ensure that your application has been initialized.

Procedure:

To change parameter bindings by using offsets:

1. Call `SQLBindParameter()` as you had been to bind the parameters.

   The first set of bound parameter buffer addresses and the corresponding length/indicator buffer addresses will act as a template. The application will then move this template to different memory locations using the offset.
2. Call `SQLExecute()` or `SQLExecDirect()` as you had been to execute the statement.

   The values stored in the bound addresses will be used.
3. Initialize a variable to hold the memory offset value.

   The statement attribute SQL_ATTR_PARAM_BIND_OFFSET_PTR points to the address of an SQLINTEGER buffer where the offset will be stored. This address must remain valid until the cursor is closed.

This extra level of indirection enables the use of a single memory variable to store the offset for multiple sets of parameter buffers on different statement handles. The application need only set this one memory variable and all of the offsets will be changed.

4. Store an offset value (number of bytes) in the memory location pointed to by the statement attribute set in the previous step.

   The offset value is always added to the memory location of the originally bound values. This sum must point to a valid memory address.
5. Call `SQLExecute()` or `SQLExecDirect()` again. CLI will add the offset specified above to the locations used in the original call to `SQLBindParameter()` to determine where the parameters to be used are stored in memory.
6. Repeat steps 4 and 5 above as required.

Related concepts:
- "Cursors in CLI Applications" on page 75

Related tasks:
- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Binding Parameter Markers in CLI Applications with Column-Wise Array Input" on page 95
- "Binding Parameter Markers in CLI Applications with Row-Wise Array Input" on page 96

Related reference:
- "C Data Types for CLI Applications" on page 54
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Array Output

Column Binding in CLI Applications

Columns may be bound to:
- Application storage

SQLBindCol() is used to bind application storage to the column. Data will be transferred from the server to the application at fetch time. Length of the available data to return is also set.

- LOB locators

  SQLBindCol() is used to bind LOB locators to the column. Only the LOB locator (4 bytes) will be transferred from the server to the application at fetch time.

  Once an application receives a locator it can be used in SQLGetSubString(), SQLGetPosition(), SQLGetLength(), or as the value of a parameter marker in another SQL statement. SQLGetSubString() can either return another locator, or the data itself. All locators remain valid until the end of the transaction in which they were created (even when the cursor moves to another row), or until it is freed using the FREE LOCATOR statement.
- Lob file references

  SQLBindFileToCol() is used to bind a file to a LOB column. DB2 CLI will write the data directly to a file, and update the *StringLength* and *IndicatorValue* buffers specified on SQLBindFileToCol().

  If the data value for the column is NULL and SQLBindFileToCol() was used, then *IndicatorValue* will be set to SQL_NULL_DATA and *StringLength* to 0.

The number of columns in a result set can be determined by calling SQLNumResultCols() or by calling SQLColAttribute() with the *DescType* argument set to SQL_COLUMN_COUNT.

The application can query the attributes (such as data type and length) of the column by first calling SQLDescribeCol() or SQLColAttribute(). This information can then be used to allocate a storage location of the correct data type and length, to indicate data conversion to another data type, or in the case of LOB data types, optionally return a locator.

An application can choose not to bind every column, or even not to bind any columns. Data in any of the columns can also be retrieved using SQLGetData() after the bound columns have been fetched for the current row. It is usually more efficient to bind application variables or file references to result sets than to use SQLGetData(). When the data is in a LOB column, LOB functions are preferable to SQLGetData() . Use SQLGetData() when the data value is large variable-length data that:
- must be received in pieces, or
- may not need to be retrieved.

Instead of multiple calls to SQLBindCol(), DB2 CLI also supports column binding offsets. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLFetch()` or `SQLFetchScroll()`. This can only be used with row wise binding, but will work whether the application retrieves a single row or multiple rows at a time.

When binding any variable length column, DB2 CLI will be able to write *StrLen_or_IndPtr* and *TargetValuePtr* in one operation if they are allocated contiguously. For example:

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

The most recent bind column function call determines the type of binding that is in effect.

Related concepts:

- "LOB Locators in CLI Applications" on page 118

Related tasks:

- "Changing Column Bindings in a CLI Application with Column Binding Offsets" on page 107

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLBindFileToCol Function (CLI) - Bind LOB File Reference to LOB Column" in the *CLI Guide and Reference, Volume 2*
- "SQLDescribeCol Function (CLI) - Return a Set of Attributes for a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "FREE LOCATOR statement" in the *SQL Reference, Volume 2*

Result Set Retrieval into Arrays in CLI Applications

One of the most common tasks performed by an application is to issue a query statement, and then fetch each row of the result set into application variables that have been bound using `SQLBindCol()`. If the application requires that each column or each row of the result set be stored in an array, each fetch must be followed by either a data copy operation or a new set of `SQLBindCol()` calls to assign new storage areas for the next fetch.

Alternatively, applications can eliminate the overhead of extra data copies or extra `SQLBindCol()` calls by retrieving multiple rows of data (called a rowset) at one time into an array.

Note: A third method of reducing overhead, which can be used on its own or with arrays, is to specify a binding offset. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLFetch()` or `SQLFetchScroll()`. This can only be used with row offset binding.

When retrieving a result set into an array, `SQLBindCol()` is also used to assign storage for application array variables. By default, the binding of rows is in column-wise fashion: this is similar to using `SQLBindParameter()` to bind arrays of input parameter values. Figure 6 is a logical view of column-wise binding.

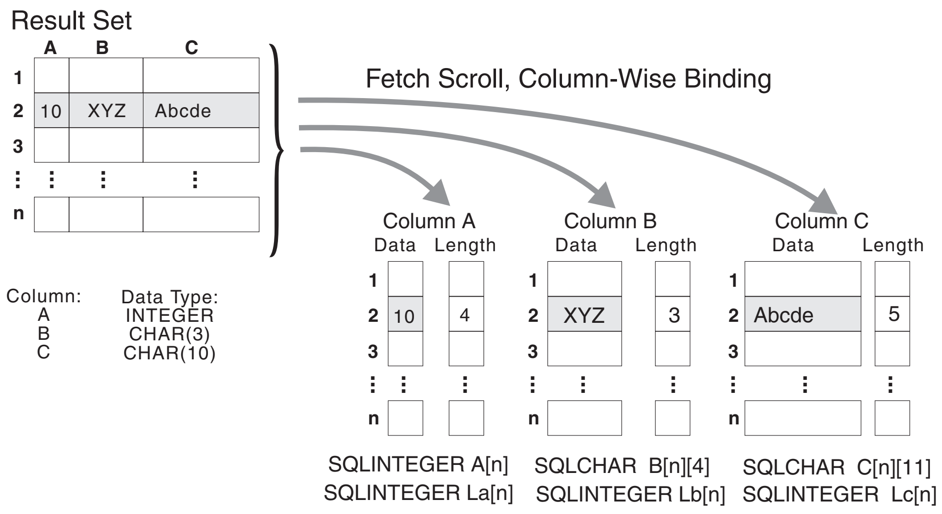


Figure 6. Column-Wise Binding

The application can also do row-wise binding which associates an entire row of the result set with a structure. In this case the rowset is retrieved into an array of structures, each of which holds the data in one row and the associated length fields. Figure 7 on page 104 gives a pictorial view of row-wise binding.

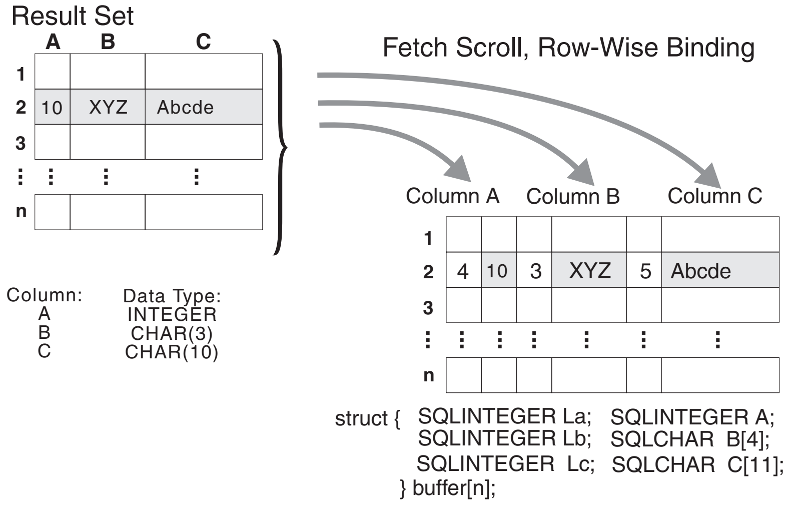

Result Set
A
B
C
1
2
10
XYZ
Abcde
3
n
Column:
A
B
C
Data Type:
INTEGER
CHAR(3)
CHAR(10)
Fetch Scroll, Row-Wise Binding
Column A
Column B
Column C
1
2
4
10
3
XYZ
5
Abcde
3
n
struct { SQLINTEGER La; SQLINTEGER A;
SQLINTEGER Lb; SQLCHAR B[4];
SQLINTEGER Lc; SQLCHAR C[11];
} buffer[n];

Figure 7. Row-wise binding

Related tasks:

- "Retrieving Array Data in CLI Applications Using Column-wise Binding" on page 104
- "Retrieving Array Data in CLI Applications Using Row-wise Binding" on page 106
- "Changing Column Bindings in a CLI Application with Column Binding Offsets" on page 107

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*

Retrieving Array Data in CLI Applications Using Column-wise Binding

When retrieving data, you may want to retrieve more than one row at a time and store the data in an array. Instead of fetching and copying each row of data into an array, or binding to new storage areas, you can retrieve multiple rows of data at once using column-wise binding. Column-wise binding is the default row-binding method whereby each data value and its length is stored in an array.

Prerequisites:

Before using column-wise binding to retrieve data into arrays, ensure you have initialized your CLI application.

Procedure:

To retrieve data using column-wise binding:

1. Allocate an array of the appropriate data type for each column data value. This array will hold the retrieved data value.
2. Allocate an array of SQLINTEGER for each column. Each array will store the length of each column's data value.
3. Specify that column-wise array retrieval will be used by setting the SQL_ATTR_ROW_BIND_TYPE statement attribute to SQL_BIND_BY_COLUMN using `SQLSetStmtAttr()`.
4. Specify the number of rows that will be retrieved by setting the SQL_ATTR_ROW_ARRAY_SIZE statement attribute using `SQLSetStmtAttr()`.

   When the value of the SQL_ATTR_ROW_ARRAY_SIZE attribute is greater than 1, DB2 CLI treats the deferred output data pointer and length pointer as pointers to arrays of data and length rather than to one single element of data and length of a result set column.
5. Prepare and execute the SQL statement used to retrieve the data.
6. Bind each array to its column by calling `SQLBindCol()` for each column.
7. Retrieve the data by calling `SQLFetch()` or `SQLFetchScroll()`.

   When returning data, DB2 CLI uses the maximum buffer size argument (*BufferLength*) of `SQLBindCol()` to determine where to store successive rows of data in the array. The number of bytes available for return for each element is stored in the deferred length array. If the number of rows in the result set is greater than the SQL_ATTR_ROW_ARRAY_SIZE attribute value, multiple calls to `SQLFetchScroll()` are required to retrieve all the rows.

Related concepts:

- "Result Set Retrieval into Arrays in CLI Applications" on page 102

Related tasks:

- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Retrieving Array Data in CLI Applications Using Row-wise Binding" on page 106

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*

- “SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns” in the *CLI Guide and Reference, Volume 2*
- “SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement” in the *CLI Guide and Reference, Volume 2*
- “Statement Attributes (CLI) List” in the *CLI Guide and Reference, Volume 2*

Related samples:
- “tbread.c -- How to read data from tables (CLI)”

Retrieving Array Data in CLI Applications Using Row-wise Binding

When retrieving data, you may want to retrieve more than one row at a time and store the data in an array. Instead of fetching and copying each row of data into an array, or binding to new storage areas, you can retrieve multiple rows of data using row-wise binding. Row-wise binding associates an entire row of the result set with a structure. The rowset is retrieved into an array of structures, each of which holds the data in one row and the associated length fields.

Prerequisites:

Before using row-wise binding to retrieve data into arrays, ensure you have initialized your CLI application.

Procedure:

To retrieve data using row-wise binding:

1. Allocate an array of structures of size equal to the number of rows to be retrieved, where each element of the structure is composed of each row’s data value and each data value’s length.

   For example, if each row of the result set consisted of Column A of type INTEGER, Column B of type CHAR(3), and Column C of type CHAR(10), then you would allocate the following structure, where n represents the number of rows in the result set:

   ```
   struct { SQLINTEGER La; SQLINTEGER A;
            SQLINTEGER Lb; SQLCHAR B[4];
            SQLINTEGER Lc; SQLCHAR C[11];
          } buffer[n];
   ```
2. Specify that row-wise array retrieval will be used by setting the SQL_ATTR_ROW_BIND_TYPE statement attribute, using `SQLSetStmtAttr()` to the size of the structure to which the result columns will be bound.
3. Specify the number of rows that will be retrieved by setting the SQL_ATTR_ROW_ARRAY_SIZE statement attribute using `SQLSetStmtAttr()`.

4. Prepare and execute the SQL statement used to retrieve the data.
5. Bind each structure to the row by calling `SQLBindCol()` for each column of the row.

   DB2 CLI treats the deferred output data pointer of `SQLBindCol()` as the address of the data field for the column in the first element of the array of structures. The deferred output length pointer is treated as the address of the associated length field of the column.
6. Retrieve the data by calling `SQLFetchScroll()`.

   When returning data, DB2 CLI uses the structure size provided with the SQL_ATTR_ROW_BIND_TYPE statement attribute to determine where to store successive rows in the array of structures.

Related concepts:

- "Result Set Retrieval into Arrays in CLI Applications" on page 102

Related tasks:

- "Initializing CLI Applications" on page 24
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Retrieving Array Data in CLI Applications Using Column-wise Binding" on page 104

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "tbread.c -- How to read data from tables (CLI)"

Changing Column Bindings in a CLI Application with Column Binding Offsets

When an application needs to change bindings (for a subsequent fetch, for example) it can call `SQLBindCol()` a second time. This will change the buffer address and length/indicator pointer used. Instead of multiple calls to `SQLBindCol()`, DB2 CLI supports column binding offsets. Rather than re-binding each time, an offset can be used to specify new buffer and length/indicator addresses which will be used in a subsequent call to `SQLFetch()` or `SQLFetchScroll()`.

Prerequisites:

Before using column binding offsets to change result set bindings, ensure you have initialized your CLI application.

Restrictions:

This method can only be used with row-wise binding, but will work whether the application retrieves a single row or multiple rows at a time.

Procedure:

To change result set bindings using column binding offsets:

1. Call `SQLBindCol()` as usual to bind the result set. The first set of bound data buffer and length/indicator buffer addresses will act as a template. The application will then move this template to different memory locations using the offset.
2. Call `SQLFetch()` or `SQLFetchScroll()` as usual to fetch the data. The data returned will be stored in the locations bound above.
3. Set up a variable to hold the memory offset value.

   The statement attribute SQL_ATTR_ROW_BIND_OFFSET_PTR points to the address of an SQLINTEGER buffer where the offset will be stored. This address must remain valid until the cursor is closed.

   This extra level of indirection enables the use of a single memory variable to store the offset for multiple sets of bindings on different statement handles. The application need only set this one memory variable and all of the offsets will be changed.
4. Store an offset value (number of bytes) in the memory location pointed to by the statement attribute set in the previous step.

   The offset value is always added to the memory location of the originally bound values. This sum must point to a valid memory address with sufficient space to hold the next set of data.
5. Call `SQLFetch()` or `SQLFetchScroll()` again. CLI will add the offset specified above to the locations used in the original call to `SQLBindCol()`. This will determine where in memory to store the results.
6. Repeat steps 4 and 5 above as required.

Related concepts:

- "Column Binding in CLI Applications" on page 100

Related tasks:

- "Initializing CLI Applications" on page 24

- “Retrieving Array Data in CLI Applications Using Row-wise Binding” on page 106

Related reference:

- “SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator” in the *CLI Guide and Reference, Volume 2*
- “SQLFetch Function (CLI) - Fetch Next Row” in the *CLI Guide and Reference, Volume 2*
- “SQLFetchScroll Function (CLI) - Fetch Rowset and Return Data for All Bound Columns” in the *CLI Guide and Reference, Volume 2*
- “Statement Attributes (CLI) List” in the *CLI Guide and Reference, Volume 2*

Chapter 7. Working with Large Amounts of Data

Specifying Parameter Values at Execute Time for Long Data Manipulation in CLI Applications

When manipulating long data, it may not be feasible for the application to load the entire parameter data value into storage at the time the statement is executed, or when the data is fetched from the database. A method has been provided to allow the application to handle the data in a piecemeal fashion. The technique of sending long data in pieces is called *specifying parameter values at execute time*. It can also be used to specify values for fixed size non-character data types such as integers.

Prerequisites:

Before specifying parameter values at execute time, ensure you have initialized your CLI application.

Restrictions:

While the data-at-execution flow is in progress, the only DB2 CLI functions the application can call are:

- `SQLParamData()` and `SQLPutData()` as given in the sequence below.
- The `SQLCancel()` function which is used to cancel the flow and force an exit from the loops described below without executing the SQL statement.
- The `SQLGetDiagRec()` function.

Procedure:

A data-at-execute parameter is a bound parameter for which a value is prompted at execution time instead of stored in memory before `SQLExecute()` or `SQLExecDirect()` is called. To indicate such a parameter on an `SQLBindParameter()` call:

1. Set the input data length pointer to point to a variable that, at execute time, will contain the value SQL_DATA_AT_EXEC. For example:

   ```
   /* dtlob.c */
   /* ... */
   SQLINTEGER      blobInd ;
   /* ... */
   blobInd = SQL_DATA_AT_EXEC;
   sqlrc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_BINARY,
                            SQL_BLOB, BUFSIZ, 0, (SQLPOINTER)inputParam,
                            BUFSIZ, &blobInd);
   ```
2. If there is more than one data-at-execute parameter, set each input data pointer argument to some value that it will recognize as uniquely identifying the field in question.
3. If there are any data-at-execute parameters when the application calls `SQLExecDirect()` or `SQLExecute()`, the call returns with SQL_NEED_DATA to prompt the application to supply values for these parameters. The application responds with the subsequent steps.
4. Call `SQLParamData()` to conceptually advance to the first such parameter. `SQLParamData()` returns SQL_NEED_DATA and provides the contents of the input data pointer argument specified on the associated `SQLBindParameter()` call to help identify the information required.
5. Pass the actual data for the parameter by calling `SQLPutData()`. Long data can be sent in pieces by calling `SQLPutData()` repeatedly.
6. Call `SQLParamData()` again after providing the entire data for this data-at-execute parameter.
7. If more data-at-execute parameters exist, `SQLParamData()` again returns SQL_NEED_DATA and the application repeats steps 4 and 5 above.

For example:

```
/* dtlob.c */
/* ... */
    else
    {   sqlrc = SQLParamData( hstmt, (SQLPOINTER *) &valuePtr);
        /* ... */
        while ( sqlrc == SQL_NEED_DATA)
        {   /*
            if more than 1 parms used DATA_AT_EXEC then valuePtr would
            have to be checked to determine which param needed data
            */
            while ( feof( pFile ) == 0 )
            {   n = fread( buffer, sizeof(char), BUFSIZ, pFile);
                sqlrc = SQLPutData(hstmt, buffer, n);
                STMT_HANDLE_CHECK( hstmt, sqlrc);
```

```
                    fileSize = fileSize + n;
                    if ( fileSize > 102400u)
                    {   /* BLOB column defined as 100K MAX */
                        /* ... */
                        break;
                    }
                }
                /* ... */
                sqlrc = SQLParamData( hstmt, (SQLPOINTER *) &valuePtr);
                /* ... */
            }
        }
```

When all data-at-execute parameters have been assigned values, `SQLParamData()` completes execution of the SQL statement and returns a return value and diagnostics as the original `SQLExecDirect()` or `SQLExecute()` would have produced.

Related tasks:

- “Initializing CLI Applications” on page 24

Related reference:

- “SQLCancel Function (CLI) - Cancel Statement” in the *CLI Guide and Reference, Volume 2*
- “SQLGetDiagRec Function (CLI) - Get Multiple Fields Settings of Diagnostic Record” in the *CLI Guide and Reference, Volume 2*
- “SQLParamData Function (CLI) - Get Next Parameter For Which A Data Value Is Needed” in the *CLI Guide and Reference, Volume 2*
- “SQLPutData Function (CLI) - Passing Data Value for A Parameter” in the *CLI Guide and Reference, Volume 2*
- “SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator” in the *CLI Guide and Reference, Volume 2*

Related samples:

- “dtlob.c -- How to read and write LOB data (CLI)”

Data Retrieval in Pieces in CLI Applications

Typically, an application may choose to allocate the maximum memory the column value could occupy and bind it via `SQLBindCol()`, based on information about a column in the result set (obtained via a call to `SQLDescribeCol()`, for example, or prior knowledge). However, in the case of character and binary data, the column can be arbitrarily long. If the length of the column value exceeds the length of the buffer the application can allocate or afford to allocate, a feature of SQLGetData() lets the application use repeated calls to obtain in sequence the value of a single column in more manageable pieces.

Basically, as shown in the left branch of the flow diagrammed in Figure 8 on page 115, a call to SQLGetData() returns SQL_SUCCESS_WITH_INFO (with SQLSTATE 01004) to indicate more data exists for this column. SQLGetData() is called repeatedly to get the remaining pieces of data until it returns SQL_SUCCESS, signifying that the entire data has been retrieved for this column. For example:

```
/* dtlob.c */
/* ... */
sqlrc = SQLGetData(hstmt, 1, SQL_C_BINARY, (SQLPOINTER) buffer,
                BUFSIZ, &bufInd);
/* ... */
while( sqlrc == SQL_SUCCESS_WITH_INFO || sqlrc == SQL_SUCCESS )
{   if ( bufInd > BUFSIZ)  /* full buffer */
    {   fwrite( buffer, sizeof(char), BUFSIZ, pFile);
    }
    else /* partial buffer on last GetData */
    {   fwrite( buffer, sizeof(char), bufInd, pFile);
    }

    sqlrc = SQLGetData( hstmt, 1, SQL_C_BINARY, (SQLPOINTER)buffer,
                        BUFSIZ, &bufInd);
    /* ... */
}
```

The function SQLGetSubString() can also be used to retrieve a specific portion of a large object value. For other alternative methods to retrieve long data, refer to the documentation on large object usage.

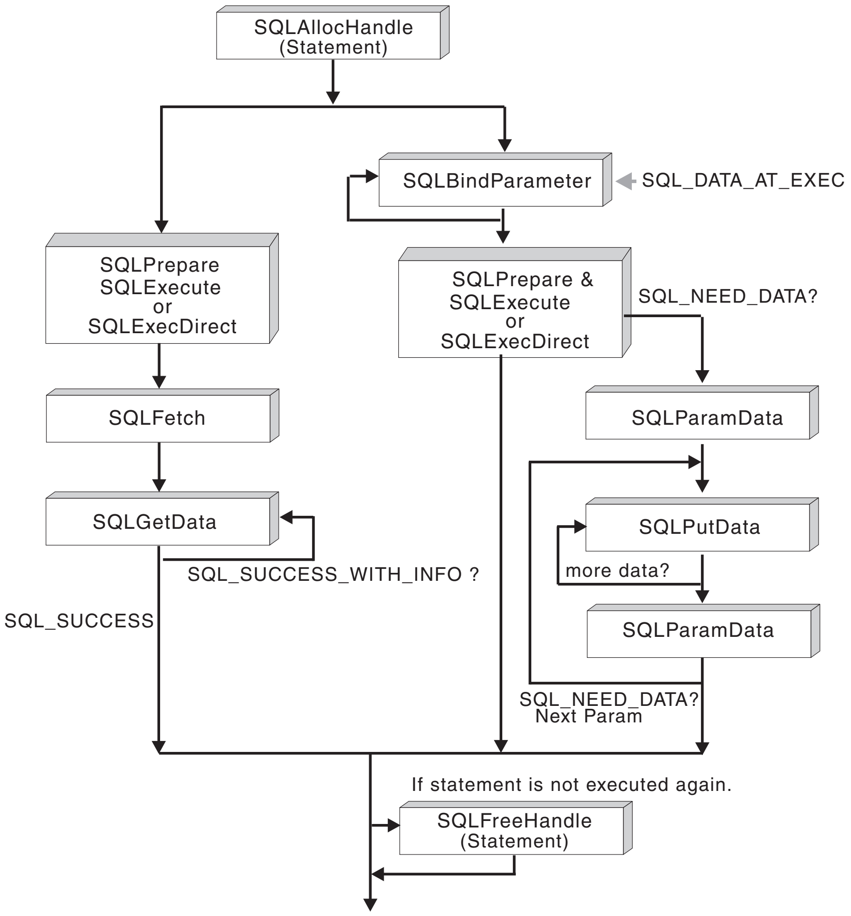

SQLAllocHandle (Statement)
SQLBindParameter
SQL_DATA_AT_EXEC
SQLPrepare SQLExecute or SQLExecDirect
SQLPrepare & SQLExecute or SQLExecDirect
SQL_NEED_DATA?
SQLFetch
SQLParamData
SQLGetData
SQLPutData
SQL_SUCCESS_WITH_INFO ?
more data?
SQL_SUCCESS
SQLParamData
SQL_NEED_DATA? Next Param
If statement is not executed again.
SQLFreeHandle (Statement)

Figure 8. Piecewise Input and Retrieval

Related concepts:

- "Large Object Usage in CLI Applications" on page 116

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLDescribeCol Function (CLI) - Return a Set of Attributes for a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLGetSubString Function (CLI) - Retrieve Portion of A String Value" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Related samples:

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

Large Object Usage in CLI Applications

The term *large object* and the generic acronym *LOB* are used to refer to any type of large object. There are three LOB data types: Binary Large Object (BLOB), Character Large Object (CLOB), and Double-Byte Character Large Object (DBCLOB). These LOB data types are represented symbolically as SQL_BLOB, SQL_CLOB, SQL_DBCLOB respectively. The LOB symbolic constants can be specified or returned on any of the DB2 CLI functions that take in or return an SQL data type argument (such as `SQLBindParameter()`, `SQLDescribeCol()`).

Since LOB values can be very large, transfer of data using the piecewise sequential method provided by `SQLGetData()` and `SQLPutData()` can be quite time consuming. Applications dealing with such data will often do so in random access segments using LOB locators or via direct file input and output.

DB2 Version 5 and later on UNIX® and Windows provide LOB support, as do DB2 servers for OS/390® Version 6 and later. To determine if any of the LOB functions are supported for the current server, call `SQLGetFunctions()` with the appropriate function name argument value, or `SQLGetTypeInfo()` with the particular LOB data type.

Figure 9 on page 117 shows the retrieval of a character LOB (CLOB).

- The left hand side shows a locator being used to extract a character string from the CLOB, without having to transfer the entire CLOB to an application buffer.

  A LOB locator is fetched, which is then used as an input parameter to search the CLOB for a substring, the substring is then retrieved.
- The right hand side shows how the CLOB can be fetched directly into a file.

  The file is first bound to the CLOB column, and when the row is fetched, the entire CLOB value is transferred directly to a file.

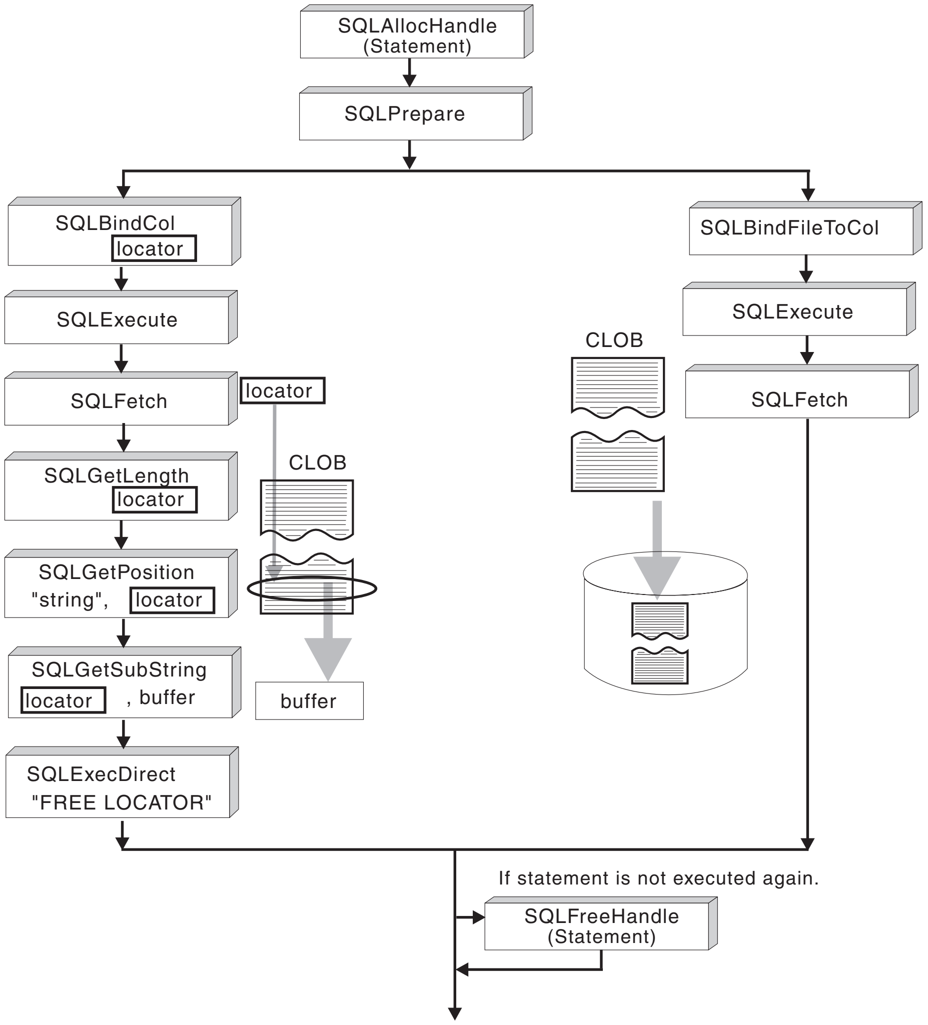


Figure 9. Fetching CLOB Data

Related concepts:

- "Data Retrieval in Pieces in CLI Applications" on page 113
- "LOB Locators in CLI Applications" on page 118

Related reference:

- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLGetFunctions Function (CLI) - Get Functions" in the *CLI Guide and Reference, Volume 2*

- "SQLGetTypeInfo Function (CLI) - Get Data Type Information" in the *CLI Guide and Reference, Volume 2*
- "SQLPutData Function (CLI) - Passing Data Value for A Parameter" in the *CLI Guide and Reference, Volume 2*

LOB Locators in CLI Applications

There are many cases where an application needs to select a large object value and operate on pieces of it, but does not need or want the entire value to be transferred from the database server into application memory. In these cases, the application can reference an individual LOB value via a large object locator (LOB locator).

A LOB locator is a token value, defined as type SQLINTEGER, that allows for efficient random access of a large object. When a LOB locator is used, the server performs the query and instead of placing the value of the LOB column in the result set, it updates the LOB locator with an integer that corresponds to the value of the LOB. When the application later requests the result, the application then passes the locator to the server and the server returns the LOB result.

A LOB locator is not stored in the database. It refers to a LOB value during a transaction, and does not persist beyond the transaction in which it was created. It is a simple token value created to reference a single large object *value*, and not a column in a row. There is no operation that could be performed on a locator that would have an effect on the original LOB value stored in the row.

Each of the three LOB locator types has its own C data type (SQL_C_BLOB_LOCATOR, SQL_C_CLOB_LOCATOR, SQL_C_DBCLOB_LOCATOR). These types are used to enable transfer of LOB locator values to and from the database server.

Locators are implicitly allocated by:
- Fetching a bound LOB column to the appropriate C locator type.
- Calling `SQLGetSubString()` and specifying that the substring be retrieved as a locator.
- Calling `SQLGetData()` on an unbound LOB column and specifying the appropriate C locator type. The C locator type must match the LOB column type or an error will occur.

LOB locators also provide an efficient method of moving data from one column of a table in a database to another column (of the same or different table) without having to pull the data first into application memory and then sending it back to the server. For example, the following INSERT statement inserts a LOB value that is a concatenation of 2 LOB values as represented by their locators:

```
INSERT INTO lobtable values (CAST ? AS CLOB(4k) || CAST ? AS CLOB(5k))
```

Differences between Regular Data Types and LOB Locators:

LOB locators can in general be treated as any other data type, but there are some important differences:

- Locators are generated at the server when a row is fetched and a LOB locator C data type is specified on `SQLBindCol()`, or when `SQLGetSubString()` is called to define a locator on a portion of another LOB. Only the locator is transferred to the application.
- The value of the locator is only valid within the current transaction. You cannot store a locator value and use it beyond the current transaction, even if the cursor used to fetch the LOB locator has the WITH HOLD attribute.
- A locator can also be freed before the end of the transaction with the FREE LOCATOR statement.
- Once a locator is received, the application can use `SQLGetSubString()`, to either receive a portion of the LOB value, or to generate another locator representing the sub-string. The locator value can also be used as input for a parameter marker (using `SQLBindParameter()`).

  A LOB locator is not a pointer to a database position, but rather it is a reference to a LOB value: a snapshot of that LOB value. There is no association between the current position of the cursor and the row from which the LOB value was extracted. This means that even after the cursor has moved to a different row, the LOB locator (and thus the value that it represents) can still be referenced.
- `SQLGetPosition()` and `SQLGetLength()` can be used with `SQLGetSubString()` to define the sub-string.

For a given LOB column in the result set, the binding can be to a:
- storage buffer for holding the entire LOB data value,
- LOB locator, or
- LOB file reference (using `SQLBindFileToCol()`).

Related concepts:
- "Parameter Marker Binding in CLI Applications" on page 33
- "Large Object Usage in CLI Applications" on page 116

Related tasks:
- "Fetching LOB Data with LOB Locators in CLI Applications" on page 120

Related reference:

- "C Data Types for CLI Applications" on page 54
- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLGetLength Function (CLI) - Retrieve Length of A String Value" in the *CLI Guide and Reference, Volume 2*
- "SQLGetPosition Function (CLI) - Return Starting Position of String" in the *CLI Guide and Reference, Volume 2*
- "SQLGetSubString Function (CLI) - Retrieve Portion of A String Value" in the *CLI Guide and Reference, Volume 2*

Fetching LOB Data with LOB Locators in CLI Applications

The following are typical steps for fetching LOB data using a LOB locator. The examples shown in each step illustrate how using a locator to retrieve CLOB data allows a character string to be extracted from the CLOB, without having to transfer the entire CLOB to an application buffer. The LOB locator is fetched and then used as an input parameter to search the CLOB for a substring. This substring is then retrieved.

Prerequisites:

Before fetching LOB data with LOB locators, ensure you have initialized your CLI application.

Procedure:

To fetch LOB data using LOB locators:

1. Retrieve a LOB locator into an application variable using the `SQLBindCol()` or `SQLGetData()` functions. For example:
   ```
   SQLINTEGER  clobLoc ;
   SQLINTEGER  pcbValue ;

   /* ... */
   sqlrc = SQLBindCol( hstmtClobFetch, 1, SQL_C_CLOB_LOCATOR,
                       &clobLoc, 0, &pcbValue);
   ```
2. Fetch the locator using `SQLFetch()`:
   ```
   sqlrc = SQLFetch( hstmtClobFetch );
   ```
3. Call `SQLGetLength()` to get the length of a string that is represented by a LOB locator. For example:
   ```
   sqlrc = SQLGetLength( hstmtLocUse, SQL_C_CLOB_LOCATOR,
                         clobLoc, &clobLen, &ind ) ;
   ```

4. Call SQLGetPosition() to get the position of a search string within a source string where the source string is represented by a LOB locator. The search string can also be represented by a LOB locator. For example:

```
sqlrc = SQLGetPosition( hstmtLocUse,
                        SQL_C_CLOB_LOCATOR,
                        clobLoc,
                        0,
                        ( SQLCHAR * ) "Interests",
                        strlen( "Interests"),
                        1,
                        &clobPiecePos,
                        &ind ) ;
```

5. Call SQLGetSubString() to retrieve the substring. For example:

```
sqlrc = SQLGetSubString( hstmtLocUse,
                         SQL_C_CLOB_LOCATOR,
                         clobLoc,
                         clobPiecePos,
                         clobLen - clobPiecePos,
                         SQL_C_CHAR,
                         buffer,
                         clobLen - clobPiecePos + 1,
                         &clobPieceLen,
                         &ind ) ;
```

6. Free the locator. All LOB locators are implicitly freed when a transaction ends. The locator can be explicitly freed before the end of a transaction by executing the FREE LOCATOR statement.

   Although this statement cannot be prepared dynamically, DB2 CLI will accept it as a valid statement on SQLPrepare() and SQLExecDirect(). The application uses SQLBindParameter() with the SQL data type argument set to the appropriate SQL and C symbolic data types. For example,

```
sqlrc = SQLSetParam( hstmtLocFree,
                     1,
                     SQL_C_CLOB_LOCATOR,
                     SQL_CLOB_LOCATOR,
                     0,
                     0,
                     &clobLoc,
                     NULL ) ;

/* ... */
sqlrc = SQLExecDirect( hstmtLocFree, stmtLocFree, SQL_NTS ) ;
```

Related concepts:

- "LOB Locators in CLI Applications" on page 118

Related tasks:

- "Initializing CLI Applications" on page 24

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLFetch Function (CLI) - Fetch Next Row" in the *CLI Guide and Reference, Volume 2*
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*
- "SQLGetLength Function (CLI) - Retrieve Length of A String Value" in the *CLI Guide and Reference, Volume 2*
- "SQLGetPosition Function (CLI) - Return Starting Position of String" in the *CLI Guide and Reference, Volume 2*
- "SQLGetSubString Function (CLI) - Retrieve Portion of A String Value" in the *CLI Guide and Reference, Volume 2*
- "FREE LOCATOR statement" in the *SQL Reference, Volume 2*

Related samples:

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

Direct File Input and Output for LOB Handling in CLI Applications

As an alternative to using LOB locators, if an application requires the entire LOB column value, it can request direct file input and output for LOBs. Database queries, updates, and inserts may involve transfer of single LOB column values into and from files. The two DB2 CLI LOB file access functions are:

`SQLBindFileToCol()`

Binds (associates) a LOB column in a result set with a file name.

Example:

```
SQLUINTEGER     fileOption = SQL_FILE_OVERWRITE;
SQLINTEGER      fileInd = 0;
SQLSMALLINT     fileNameLength = 14;
/* ... */
SQLCHAR         fileName[14] = "";

/* ... */
rc = SQLBindFileToCol(hstmt, 1, fileName, &fileNameLength,
                      &fileOption, 14, NULL, &fileInd);
```

`SQLBindFileToParam()`

Binds (associates) a LOB parameter marker with a file name.

Example:

```
SQLUINTEGER     fileOption = SQL_FILE_OVERWRITE;
SQLINTEGER      fileInd = 0;
SQLSMALLINT     fileNameLength = 14;
/* ... */
SQLCHAR         fileName[14] = "";

/* ... */

rc = SQLBindFileToParam(hstmt, 3, SQL_BLOB, fileName,
        &fileNameLength, &fileOption, 14, &fileInd);
```

The file name is either the complete path name of the file (which is recommended), or a relative file name. If a relative file name is provided, it is appended to the current path (of the operating environment) of the client process. On execute or fetch, data transfer to and from the file would take place, in a similar way to that of bound application variables. A file options argument associated with these 2 functions indicates how the files are to be handled at time of transfer.

Use of `SQLBindFileToParam()` is more efficient than the sequential input of data segments using `SQLPutData()`, since `SQLPutData()` essentially puts the input segments into a temporary file and then uses the `SQLBindFileToParam()` technique to send the LOB data value to the server. Applications should take advantage of `SQLBindFileToParam()` instead of using `SQLPutData()`.

Note: DB2 CLI uses a temporary file when inserting LOB data in pieces. If the data originates in a file, the use of a temporary file can be avoided by using `SQLBindFileToParam()`. Call `SQLGetFunctions()` to query if support is provided for `SQLBindFileToParam()`, since `SQLBindFileToParam()` is not supported against servers that do not support LOBs.

Related concepts:

- "Large Object Usage in CLI Applications" on page 116
- "LOB Locators in CLI Applications" on page 118

Related reference:

- "SQLBindFileToCol Function (CLI) - Bind LOB File Reference to LOB Column" in the *CLI Guide and Reference, Volume 2*
- "SQLBindFileToParam Function (CLI) - Bind LOB File Reference to LOB Parameter" in the *CLI Guide and Reference, Volume 2*
- "SQLGetFunctions Function (CLI) - Get Functions" in the *CLI Guide and Reference, Volume 2*
- "SQLPutData Function (CLI) - Passing Data Value for A Parameter" in the *CLI Guide and Reference, Volume 2*

Related samples:

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

LOB Usage in ODBC Applications

Existing ODBC-compliant applications use SQL_LONGVARCHAR and SQL_LONGVARBINARY instead of the DB2 BLOB and CLOB data types. You can still access LOB columns from these ODBC-compliant applications by setting the LONGDATACOMPAT configuration keyword in the initialization file, or setting the SQL_ATTR_LONGDATA_COMPAT connection attribute using `SQLSetConnectAttr()`. Once this is done, DB2 CLI will map the ODBC long data types to the DB2 LOB data types. The LOBMAXCOLUMNSIZE configuration keyword allows you to override the default COLUMN_SIZE for LOB data types.

When this mapping is in effect:

- `SQLGetTypeInfo()` will return CLOB, BLOB and DBCLOB characteristics when called with SQL_LONGVARCHAR, SQL_LONGVARBINARY or SQL_LONGVARGRAPHIC.
- The following functions will return SQL_LONGVARCHAR, SQL_LONGVARBINARY or SQL_LONGVARGRAPHIC when describing CLOB, BLOB or DBCLOB data types:
  - `SQLColumns()`
  - `SQLSpecialColumns()`
  - `SQLDescribeCol()`
  - `SQLColAttribute()`
  - `SQLProcedureColumns()`
- LONG VARCHAR and LONG VARCHAR FOR BIT DATA will continue to be described as SQL_LONGVARCHAR and SQL_LONGVARBINARY.

The default setting for SQL_ATTR_LONGDATA_COMPAT is SQL_LD_COMPAT_NO; that is, mapping is not in effect.

With mapping in effect, ODBC applications can retrieve and input LOB data by using the `SQLGetData()`, `SQLPutData()` and related functions.

Related concepts:

- "Large Object Usage in CLI Applications" on page 116

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "SQLGetData Function (CLI) - Get Data From a Column" in the *CLI Guide and Reference, Volume 2*

- "SQLPutData Function (CLI) - Passing Data Value for A Parameter" in the *CLI Guide and Reference, Volume 2*
- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*
- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "LOBMAXCOLUMNSIZE CLI/ODBC Configuration Keyword" on page 323
- "LONGDATACOMPAT CLI/ODBC Configuration Keyword" on page 323

Bulk Data Manipulation

Long Data for Bulk Inserts and Updates in CLI Applications

Long data can be provided for bulk inserts and updates performed by calls to `SQLBulkOperations()`.

1. When an application binds the data using `SQLBindCol()`, the application places an application-defined value, such as the column number, in the **TargetValuePtr* buffer for data-at-execution columns. The value can be used later to identify the column.

   The application places the result of the SQL_LEN_DATA_AT_EXEC(*length*) macro in the **StrLen_or_IndPtr* buffer. If the SQL data type of the column is SQL_LONGVARBINARY, SQL_LONGVARCHAR, or a long, data source-specific data type and CLI returns "Y" for the SQL_NEED_LONG_DATA_LEN information type in `SQLGetInfo()`, *length* is the number of bytes of data to be sent for the parameter; otherwise, it must be a non-negative value and is ignored.
2. When `SQLBulkOperations()` is called, if there are data-at-execution columns, the function returns SQL_NEED_DATA and proceeds to the next event in the sequence, described in the next item. (If there are no data-at-execution columns, the process is complete.)
3. The application calls `SQLParamData()` to retrieve the address of the **TargetValuePtr* buffer for the first data-at-execution column to be processed. `SQLParamData()` returns SQL_NEED_DATA. The application retrieves the application-defined value from the **TargetValuePtr* buffer.

   Note: Although data-at-execution parameters are similar to data-at-execution columns, the value returned by `SQLParamData()` is different for each.

   Data-at-execution columns are columns in a rowset for which data will be sent with `SQLPutData()` when a row is updated or inserted with `SQLBulkOperations()`. They are bound with `SQLBindCol()`. The value returned by `SQLParamData()` is the address of the row in the **TargetValuePtr* buffer that is being processed.

4. The application calls SQLPutData() one or more times to send data for the column. More than one call is needed if all the data value cannot be returned in the **TargetValuePtr* buffer specified in SQLPutData(); note that multiple calls to SQLPutData() for the same column are allowed only when sending character C data to a column with a character, binary, or data source-specific data type or when sending binary C data to a column with a character, binary, or data source-specific data type.
5. The application calls SQLParamData() again to signal that all data has been sent for the column.
   - If there are more data-at-execution columns, SQLParamData() returns SQL_NEED_DATA and the address of the *TargetValuePtr* buffer for the next data-at-execution column to be processed. The application repeats steps 4 and 5 above.
   - If there are no more data-at-execution columns, the process is complete. If the statement was executed successfully, SQLParamData() returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO; if the execution failed, it returns SQL_ERROR. At this point, SQLParamData() can return any SQLSTATE that can be returned by SQLBulkOperations().

If the operation is canceled, or an error occurs in SQLParamData() or SQLPutData(), after SQLBulkOperations() returns SQL_NEED_DATA, and before data is sent for all data-at-execution columns, the application can call only SQLCancel(), SQLGetDiagField(), SQLGetDiagRec(), SQLGetFunctions(), SQLParamData(), or SQLPutData() for the statement or the connection associated with the statement. If it calls any other function for the statement or the connection associated with the statement, the function returns SQL_ERROR and SQLSTATE HY010 (Function sequence error).

If the application calls SQLCancel() while CLI still needs data for data-at-execution columns, CLI cancels the operation. The application can then call SQLBulkOperations() again; canceling does not affect the cursor state or the current cursor position.

Related reference:
- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "SQLGetInfo Function (CLI) - Get General Information" in the *CLI Guide and Reference, Volume 2*
- "SQLParamData Function (CLI) - Get Next Parameter For Which A Data Value Is Needed" in the *CLI Guide and Reference, Volume 2*
- "SQLPutData Function (CLI) - Passing Data Value for A Parameter" in the *CLI Guide and Reference, Volume 2*
- "SQLBulkOperations Function (CLI) - Add, Update, Delete or Fetch a Set of Rows" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Retrieving Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications

You can retrieve, or fetch, bulk data using bookmarks and the DB2 CLI `SQLBulkOperations()` function.

Prerequisites:

Before fetching bulk data using bookmarks and `SQLBulkOperations()`, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating with bookmarks.

Procedure:

To perform bulk fetches using bookmarks with `SQLBulkOperations()`:

1. Set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.
3. Set the SQL_ATTR_ROW_ARRAY_SIZE statement attribute to the number of rows you want to fetch by calling `SQLSetStmtAttr()`.
4. Call `SQLBindCol()` to bind the data you want to fetch.

   The data is bound to an array with a size equal to the value of SQL_ATTR_ROW_ARRAY_SIZE.
5. Call `SQLBindCol()` to bind column 0, the bookmark column.
6. Copy the bookmarks for rows you want to fetch into the array bound to column 0.

   Note: The size of the array pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute should either be equal to SQL_ATTR_ROW_ARRAY_SIZE, or the SQL_ATTR_ROW_STATUS_PTR statement attribute should be a null pointer.
7. Fetch the data by calling `SQLBulkOperations()` with an *Operation* argument of SQL_FETCH_BY_BOOKMARK.

   If the application has set the SQL_ATTR_ROW_STATUS_PTR statement attribute, then it can inspect this array to see the result of the operation.

Related concepts:

- “Bookmarks in CLI Applications” on page 90

Related tasks:

- “Initializing CLI Applications” on page 24
- “Inserting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications” on page 128
- “Deleting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications” on page 131
- “Updating Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications” on page 130

Related reference:

- “SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator” in the *CLI Guide and Reference, Volume 2*
- “SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement” in the *CLI Guide and Reference, Volume 2*
- “SQLBulkOperations Function (CLI) - Add, Update, Delete or Fetch a Set of Rows” in the *CLI Guide and Reference, Volume 2*
- “Statement Attributes (CLI) List” in the *CLI Guide and Reference, Volume 2*

Inserting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications

You can insert data in bulk with bookmarks using `SQLBulkOperations()`.

Prerequisites:

Before inserting bulk data with `SQLBulkOperations()`, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating with bookmarks.

Procedure:

To perform a bulk data insert using `SQLBulkOperations()`:

1. Set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.

3. Set the SQL_ATTR_ROW_ARRAY_SIZE statement attribute to the number of rows you want to insert using `SQLSetStmtAttr()`.
4. Call `SQLBindCol()` to bind the data you want to insert.

   The data is bound to an array with a size equal to the value of SQL_ATTR_ROW_ARRAY_SIZE, set in the previous step.

   Note: The size of the array pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute should either be equal to SQL_ATTR_ROW_ARRAY_SIZE or SQL_ATTR_ROW_STATUS_PTR should be a null pointer.
5. Insert the data by calling `SQLBulkOperations()` with SQL_ADD as the *Operation* argument.

   CLI will update the bound column 0 buffers with the bookmark values for the newly inserted rows. For this to occur, the application must have set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE before executing the statement.

   Note: If `SQLBulkOperations()` is called with an *Operation* argument of SQL_ADD on a cursor that contains duplicate columns, an error is returned.

Related concepts:
- "Bookmarks in CLI Applications" on page 90

Related tasks:
- "Initializing CLI Applications" on page 24
- "Retrieving Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 127
- "Deleting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 131
- "Updating Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 130

Related reference:
- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBulkOperations Function (CLI) - Add, Update, Delete or Fetch a Set of Rows" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Updating Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications

You can update data in bulk with bookmarks using `SQLBulkOperations()`.

Prerequisites:

Before updating data in bulk, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating with bookmarks.

Procedure:

To update data in bulk:

1. Set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.
3. Set the SQL_ATTR_ROW_ARRAY_SIZE statement attribute to the number of rows you want to update using `SQLSetStmtAttr()`.
4. Call `SQLBindCol()` to bind the data you want to update.

   The data is bound to an array with a size equal to the value of SQL_ATTR_ROW_ARRAY_SIZE, set in the previous step.
5. Bind the bookmark column to column 0 by calling `SQLBindCol()`.
6. Copy the bookmarks for rows that you want to update into the array bound to column 0.
7. Update the data in the bound buffers.

   Note: The size of the array pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute should either be equal to SQL_ATTR_ROW_ARRAY_SIZE or SQL_ATTR_ROW_STATUS_PTR should be a null pointer.
8. Update the data by calling `SQLBulkOperations()` with an *Operation* argument of SQL_UPDATE_BY_BOOKMARK.

   Note: If the application has set the SQL_ATTR_ROW_STATUS_PTR statement attribute, then it can inspect this array to see the result of the operation.

9. Optional: Verify that the update has occurred by calling `SQLBulkOperations()` with an *Operation* argument of SQL_FETCH_BY_BOOKMARK. This will fetch the data into the bound application buffers.

   If data has been updated, CLI changes the value in the row status array for the appropriate rows to SQL_ROW_UPDATED.

   Note: If `SQLBulkOperations()` is called with an *Operation* argument of SQL_UPDATE_BY_BOOKMARK on a cursor that contains duplicate columns, an error is returned.

Related concepts:

- "Bookmarks in CLI Applications" on page 90

Related tasks:

- "Initializing CLI Applications" on page 24
- "Inserting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 128
- "Retrieving Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 127
- "Deleting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 131

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBulkOperations Function (CLI) - Add, Update, Delete or Fetch a Set of Rows" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Deleting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications

You can use `SQLBulkOperations()` and bookmarks to delete data in bulk.

Prerequisites:

Before deleting data in bulk, ensure you have initialized your CLI application.

Restrictions:

Bookmarks in DB2 CLI do not persist across cursor close operations. This means that an application cannot use bookmarks that it has stored from a previous cursor. Instead, it has to call `SQLFetch()` or `SQLFetchScroll()` to retrieve the bookmarks before updating by bookmarks.

Procedure:

To perform bulk deletions using bookmarks and `SQLBulkOperations()`:

1. Set the SQL_ATTR_USE_BOOKMARKS statement attribute to SQL_UB_VARIABLE using `SQLSetStmtAttr()`.
2. Execute a query that returns a result set.
3. Set the SQL_ATTR_ROW_ARRAY_SIZE statement attribute to the number of rows you want to delete.
4. Bind the bookmark column to column 0 by calling `SQLBindCol()`.
5. Copy the bookmarks for the rows you want to delete into the array bound to column 0.

   Note: The size of the array pointed to by the SQL_ATTR_ROW_STATUS_PTR statement attribute should either be equal to SQL_ATTR_ROW_ARRAY_SIZE, or the SQL_ATTR_ROW_STATUS_PTR statement attribute should be a null pointer.
6. Perform the deletion by calling `SQLBulkOperations()` with an *Operation* argument of SQL_DELETE_BY_BOOKMARK.

   If the application has set the SQL_ATTR_ROW_STATUS_PTR statement attribute, then it can inspect this array to see the result of the operation.

Related concepts:

- "Bookmarks in CLI Applications" on page 90

Related tasks:

- "Initializing CLI Applications" on page 24
- "Inserting Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 128
- "Retrieving Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 127
- "Updating Bulk Data with Bookmarks using SQLBulkOperations() in CLI Applications" on page 130

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*

- “SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement” in the *CLI Guide and Reference, Volume 2*
- “SQLBulkOperations Function (CLI) - Add, Update, Delete or Fetch a Set of Rows” in the *CLI Guide and Reference, Volume 2*
- “Statement Attributes (CLI) List” in the *CLI Guide and Reference, Volume 2*

Importing Data with the CLI LOAD Utility in CLI Applications

The CLI LOAD functionality provides an interface to the IBM DB2 LOAD utility from CLI. This functionality allows you to insert data in CLI using LOAD instead of array insert. This option can yield significant performance benefits when large amounts of data need to be inserted. Because this interface invokes LOAD, the same consideration given for using LOAD should also be taken into account when using the CLI LOAD interface.

Prerequisites:

Before importing data with the CLI LOAD utility, ensure you have initialized your CLI application.

Restrictions:

- Unlike the IBM DB2 LOAD utility, the CLI LOAD utility does not load data directly from an input file. Instead, if desired, the application should retrieve the data from the input file and insert it into the appropriate application parameters that correspond to the parameter markers in the prepared statement.
- The insertion of data is non-atomic because the load utility precludes atomicity. LOAD may not be able to successfully insert all the rows passed to it. For example, if a unique key constraint is violated by a row being inserted, LOAD will not insert this row but will continue loading the remaining rows.
- A COMMIT will be issued by LOAD. Therefore, if the insertion of the data completes successfully, the LOAD and any other statements within the transaction cannot be rolled back.
- The error reporting for the CLI LOAD interface differs from that of array insert. Non-severe errors or warnings, such as errors with specific rows, will only appear in the LOAD message file.

Procedure:

To import data using the CLI LOAD utility:

1. Specify the statement attribute SQL_ATTR_USE_LOAD_API in `SQLSetStmtAttr()` with one of the following supported values:

SQL_USE_LOAD_INSERT
Use the LOAD utility to append to existing data in the table.

SQL_USE_LOAD_REPLACE
Use the LOAD utility to replace existing data in the table.

For example, the following call indicates that the CLI LOAD utility will be used to add to the existing data in the table:

```
SQLSetStmtAttr (hStmt, SQL_ATTR_USE_LOAD_API,
                (SQLPOINTER) SQL_USE_LOAD_INSERT, 0);
```

Note: When SQL_USE_LOAD_INSERT or SQL_USE_LOAD_REPLACE is set, no other CLI functions except for the following can be called until SQL_USE_LOAD_OFF is set (see Step 3 below):
- SQLBindParameter()
- SQLExecute()
- SQLExtendedBind()
- SQLParamOptions()
- SQLSetStmtAttr()

2. Create a structure of type db2LoadStruct and specify the desired load options through this structure. Set the SQL_ATTR_LOAD_INFO statement attribute to a pointer to this structure.
3. Issue `SQLExecute()` on the prepared SQL statement for the data to be inserted.
4. Call `SQLSetStmtAttr()` with SQL_USE_LOAD_OFF. This ends the processing of data using the LOAD utility. Subsequently, regular CLI array insert will be in effect until SQL_ATTR_USE_LOAD_API is set again (see Step 1).

Example:

The following program demonstrates how you can use the CLI LOAD interface:

```
/*
 *  This program demonstrates the usage of the CLI LOAD feature.  An array
 *  of rows of size "ARRAY_SIZE" will be inserted "NUM_ITERATIONS" times.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlutil.h>
#include <sql.h>
#include <sqlenv.h>
#include <db2ApiDf.h>
#include <sqlcli.h>
```

```
#include <sqlcli1.h>


#define MESSAGE_FILE "/tmp/msgfile.txt"
#define DATABASE_NAME "mydb"
#define SAMPLE_DATA "varchar data"
#define ARRAY_SIZE 10
#define NUM_ITERATIONS 3
#define TRUE 1
#define FALSE 0


/*
 *  This function will turn the CLI LOAD feature on or off.
 */

SQLRETURN setCLILoadMode( SQLHSTMT hStmt, short fStartLoad,
                          db2LoadStruct *pLoadStruct )
{
    SQLINTEGER rc = 0;

    if( fStartLoad )
    {
        rc = SQLSetStmtAttr( hStmt, SQL_ATTR_USE_LOAD_API,
                             (SQLPOINTER)SQL_USE_LOAD_INSERT, 0 );
        if( rc != SQL_SUCCESS )
        {
            goto exit;
        }
        rc = SQLSetStmtAttr( hStmt, SQL_ATTR_LOAD_INFO,
                             (SQLPOINTER)pLoadStruct, 0 );
        if( rc != SQL_SUCCESS )
        {
            goto exit;
        }
    }
    else
    {
        rc = SQLSetStmtAttr( hStmt, SQL_ATTR_USE_LOAD_API,
                             (SQLPOINTER)SQL_USE_LOAD_OFF, 0 );
        if( rc != SQL_SUCCESS )
        {
            goto exit;
        }
    }

exit:
    return( rc );
}


/*
 *  Main program function.
 */
```

```
int main(void)
{
    SQLINTEGER rc = 0;
    SQLHANDLE henv = NULL;
    SQLHANDLE hdbc = NULL;
    SQLHANDLE hstmt = NULL;
    char statementText[1000];
    char *pTempBuffer = NULL;
    SQLINTEGER iBufferSize;
    SQLINTEGER iLoop;
    char *pColumnData = NULL;
    SQLINTEGER *pColumnSizes = NULL;
    db2LoadIn *pLoadIn = NULL;
    db2LoadOut *pLoadOut = NULL;
    db2LoadStruct *pLoadStruct = NULL;
    struct sqldcol *pDataDescriptor = NULL;
    char *pMessageFile = NULL;
    SQLINTEGER iRowsRead = 0;
    SQLINTEGER iRowsSkipped = 0;
    SQLINTEGER iRowsLoaded = 0;
    SQLINTEGER iRowsRejected = 0;
    SQLINTEGER iRowsDeleted = 0;
    SQLINTEGER iRowsCommitted = 0;

/*
 *  Allocate load structures.
 *
 *  Note that the memory belonging to the db2LoadStruct structure used
 *  in setting the SQL_ATTR_LOAD_INFO attribute *MUST* be accessible
 *  by *ALL* functions that call CLI functions for the duration of the
 *  CLI LOAD.  For this reason, it is recommended that the db2LoadStruct
 *  structure and all its embedded pointers be allocated dynamically,
 *  instead of declared statically.
 */

    pLoadIn = (db2LoadIn *)malloc(sizeof(db2LoadIn));
    if (pLoadIn == NULL)
    {
        printf("Error allocating pLoadIn!\n");
        goto exit;
    }
    pLoadOut = (db2LoadOut *)malloc(sizeof(db2LoadOut));
    if (pLoadOut == NULL)
    {
        printf("Error allocating pLoadOut!\n");
        goto exit;
    }
    pLoadStruct = (db2LoadStruct *)malloc(sizeof(db2LoadStruct));
    if (pLoadStruct == NULL)
    {
        printf("Error allocating pLoadStruct!\n");
        goto exit;
    }
    pDataDescriptor = (struct sqldcol *)malloc(sizeof(struct sqldcol));
    if (pDataDescriptor == NULL)
    {
        printf("Error allocating pDataDescriptor!\n");
        goto exit;
    }
    pMessageFile = (char *)malloc(256);
    if (pMessageFile == NULL)
    {
        printf("Error allocating pMessageFile!\n");
        goto exit;
    }

/*
 *  Initialize load structures.
 */

    memset(pDataDescriptor, 0, sizeof(struct sqldcol));
    memset(pLoadIn, 0, sizeof(db2LoadIn));
    memset(pLoadOut, 0, sizeof(db2LoadOut));
    memset(pLoadStruct, 0, sizeof(db2LoadStruct));

    pLoadStruct->piSourceList = NULL;
    pLoadStruct->piLobPathList = NULL;
    pLoadStruct->piDataDescriptor = pDataDescriptor;
    pLoadStruct->piFileTypeMod = NULL;
    pLoadStruct->piTempFilesPath = NULL;
    pLoadStruct->piVendorSortWorkPaths = NULL;
    pLoadStruct->piCopyTargetList = NULL;
    pLoadStruct->piNullIndicators = NULL;
    pLoadStruct->piLoadInfoIn = pLoadIn;
    pLoadStruct->poLoadInfoOut = pLoadOut;

    pLoadIn->iRestartphase = ' ';
    pLoadIn->iNonrecoverable = SQLU_NON_RECOVERABLE_LOAD;
    pLoadIn->iStatsOpt = (char)SQLU_STATS_NONE;
    pLoadIn->iSavecount = 0;
    pLoadIn->iCpuParallelism = 0;
    pLoadIn->iDiskParallelism = 0;
    pLoadIn->iIndexingMode = 0;
    pLoadIn->iDataBufferSize = 0;

    sprintf(pMessageFile, "%s", MESSAGE_FILE);
    pLoadStruct->piLocalMsgFileName = pMessageFile;
    pDataDescriptor->dcolmeth = SQL_METH_D;

/*
 *  Allocate handles.
 */

    rc = SQLAllocHandle( SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv ) ;
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered while allocating henv!\n", rc);
        goto exit;
    }
    rc = SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc);
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered while allocating hdbc!\n", rc);
        goto exit;
    }

/*
 *  Set AUTOCOMMIT off.
 */

    rc = SQLSetConnectAttr( hdbc,
                            SQL_ATTR_AUTOCOMMIT,
                            ( void * ) SQL_AUTOCOMMIT_OFF, SQL_NTS );
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered while turning autocommit off!\n", rc);
        goto exit;
    }

/*
 *  Connect to database.
 */

    rc = SQLConnect( hdbc,
                     (SQLCHAR *)DATABASE_NAME, SQL_NTS,
                     NULL, SQL_NTS,
                     NULL, SQL_NTS );
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered during database connect!\n", rc);
        goto exit;
    }

/*
 *  Allocate statement handle.
 */

    rc = SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt ) ;
    if ( rc != SQL_SUCCESS )
    {
        printf("006rc %d\n", rc);
        goto exit;
    }

/*
 *  Drop and recreate table "mytable".
 */

    sprintf(statementText, "drop table mytable");
    printf("%s\n", statementText);
    rc = SQLExecDirect(hstmt, (SQLCHAR *)statementText, strlen(statementText));
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered while dropping table!\n", rc);
    }

    sprintf(statementText, "create table mytable (one varchar(30))");
    printf("%s\n", statementText);
    rc = SQLExecDirect(hstmt, (SQLCHAR *)statementText, strlen(statementText));
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered while creating table!\n", rc);
    }

/*
 *  Allocate a buffer to hold data to insert.
 */

    iBufferSize = strlen(SAMPLE_DATA) * ARRAY_SIZE +
                  sizeof(SQLINTEGER) * ARRAY_SIZE;

    pTempBuffer = (char *)malloc(iBufferSize);
    memset(pTempBuffer, 0, iBufferSize);

    pColumnData = pTempBuffer;
    pColumnSizes = (SQLINTEGER *)(pColumnData +
                                  strlen(SAMPLE_DATA) * ARRAY_SIZE);

/*
 *  Initialize the array of rows.
 */

    for (iLoop=0; iLoop<ARRAY_SIZE; iLoop++)
    {
        memcpy(pColumnData + iLoop * strlen(SAMPLE_DATA), SAMPLE_DATA,
               strlen((char *)SAMPLE_DATA));
        pColumnSizes[iLoop] = strlen((char *)SAMPLE_DATA);
    }

/*
 *  Prepare the INSERT statement.
 */

    sprintf(statementText, "insert into mytable values (?)");
    rc = SQLPrepare(hstmt, (SQLCHAR *)statementText, strlen(statementText));
    if ( rc != SQL_SUCCESS )
    {
      printf("rc %d encountered during prepare!\n", rc);
      goto exit;
    }

/*
 *  Set the array size.
 */

    rc = SQLParamOptions(hstmt, ARRAY_SIZE, NULL);
    if ( rc != SQL_SUCCESS )
    {
      printf("rc %d encountered while setting array size!\n", rc);
      goto exit;
    }

/*
 *  Bind parameters.
 */

    rc = SQLBindParameter(hstmt,
                          1,
                          SQL_PARAM_INPUT,
                          SQL_C_CHAR,
                          SQL_VARCHAR,
                          30,
                          0,
                          (SQLPOINTER)pColumnData,
                          strlen(SAMPLE_DATA),
                          pColumnSizes);
    if ( rc != SQL_SUCCESS )
    {
      printf("rc %d encountered while binding!\n", rc);
      goto exit;
    }

/*
 *  Turn CLI LOAD on.
 */

    rc = setCLILoadMode(hstmt, TRUE, pLoadStruct);
    if ( rc != SQL_SUCCESS )
    {
      printf("rc %d encountered while turning on CLI LOAD!\n", rc);
      goto exit;
    }

/*
 *  Now insert the data.
 */

    for (iLoop=0; iLoop<NUM_ITERATIONS; iLoop++)
    {
        printf("Inserting %d rows..\n", ARRAY_SIZE);
        rc = SQLExecute(hstmt);
        if ( rc != SQL_SUCCESS )
        {
            printf("rc %d encountered during execute!\n", rc);
            goto exit;
        }
    }

/*
 *  Turn CLI LOAD off.
 */

    rc = setCLILoadMode(hstmt, FALSE, pLoadStruct);
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered while turning off CLI/Load!\n", rc);
        goto exit;
    }

    printf("Load messages can be found in file [%s].\n", MESSAGE_FILE);
    printf("Load report :\n");
    iRowsRead = pLoadOut->oRowsRead;
    printf("Number of rows read      : %d\n", iRowsRead);
    iRowsSkipped = pLoadOut->oRowsSkipped;
    printf("Number of rows skipped   : %d\n", iRowsSkipped);
    iRowsLoaded = pLoadOut->oRowsLoaded;
    printf("Number of rows loaded    : %d\n", iRowsLoaded);
    iRowsRejected = pLoadOut->oRowsRejected;
    printf("Number of rows rejected  : %d\n", iRowsRejected);
    iRowsDeleted = pLoadOut->oRowsDeleted;
    printf("Number of rows deleted   : %d\n", iRowsDeleted);
    iRowsCommitted = pLoadOut->oRowsCommitted;
    printf("Number of rows committed : %d\n", iRowsCommitted);

exit:

/*
 *  Now clean up.
 */

    rc = SQLEndTran(SQL_HANDLE_DBC, hdbc, SQL_COMMIT);
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered during end transaction!\n", rc);
    }

    rc = SQLDisconnect(hdbc);
    if ( rc != SQL_SUCCESS )
    {
        printf("rc %d encountered during connect reset!\n", rc);
    }

    printf("Disconnected from database..\n");

    if (pTempBuffer != NULL)
    {
        free(pTempBuffer);
        pTempBuffer = NULL;
    }

    if (pLoadIn != NULL)
    {
        free(pLoadIn);
        pLoadIn = NULL;
    }

    if (pLoadOut != NULL)
    {
        free(pLoadOut);
        pLoadOut = NULL;
    }

    if (pLoadStruct != NULL)
    {
        free(pLoadStruct);
        pLoadStruct = NULL;
    }

    if (pDataDescriptor != NULL)
    {
        free(pDataDescriptor);
        pDataDescriptor = NULL;
    }

    if (pMessageFile != NULL)
    {
        free(pMessageFile);
        pMessageFile = NULL;
    }

    return(0);
}
```

Related tasks:

- "Initializing CLI Applications" on page 24
- "Binding Parameter Markers in CLI Applications with Column-Wise Array Input" on page 95
- "Binding Parameter Markers in CLI Applications with Row-Wise Array Input" on page 96

Related reference:

- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "LOAD" in the *Command Reference*
- "db2Load - Load" in the *Administrative API Reference*

Chapter 8. Stored Procedures

Calling Stored Procedures in CLI Applications

CLI applications invoke stored procedures by executing the CALL procedure SQL statement. This topic describes how to call stored procedures from CLI applications.

Prerequisites:

Before calling a stored procedure, ensure you have initialized your CLI application.

Procedure:

To call a stored procedure:

1. Declare application host variables corresponding to each of the IN, INOUT, and OUT parameters of the stored procedure. Ensure the application variable data types and lengths match the data types and lengths of the arguments in the stored procedure signature. DB2 CLI only supports calling stored procedures with all SQL types as parameter markers.
2. Initialize the IN, INOUT, and OUT parameter application variables.
3. Issue the CALL SQL statement. For example:

   ```
   SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (?)";
   ```

   For optimal performance, applications should use parameter markers for stored procedure arguments in the CALL procedure string and then bind the host variables to those parameter markers. If inbound stored procedure arguments must be specified as string literals rather than parameter markers, however, include the ODBC call escape clause delimiters { } in the CALL procedure statement. For example:

   ```
   SQLCHAR *stmt = (SQLCHAR *)"{CALL IN_PARAM (123, 'Hello World!')}";
   ```

   When string literals and the ODBC escape clause are used in a CALL procedure statement, the string literals can only be specified as IN mode stored procedure arguments. INOUT and OUT mode stored procedure arguments must still be specified using parameter markers.
4. Optional: Prepare the CALL statement by calling `SQLPrepare()`.
5. Bind each parameter of the CALL procedure statement by calling `SQLBindParameter()`.

Note: Ensure each parameter is bound correctly (to SQL_PARAM_INPUT, SQL_PARAM_OUTPUT, or SQL_PARAM_INPUT_OUTPUT), otherwise unexpected results could occur when the CALL procedure statement is executed. This would happen, for example, if an input parameter was incorrectly bound with an *InputOutputType* of SQL_PARAM_OUTPUT.

6. Execute the CALL procedure statement using `SQLExecDirect()`, or if the CALL procedure statement was prepared in step 4, `SQLExecute()`.

   Note: If an application or thread that has invoked a stored procedure is terminated before the stored procedure completes, execution of the stored procedure will also be terminated. It is important that a stored procedure contain logic to ensure that the database is in both a consistent and desirable state if the stored procedure is terminated prematurely.

7. Check the return code of `SQLExecDirect()` or `SQLExecute()` when the function has returned to determine if any errors occurred during execution of either the CALL procedure statement or the stored procedure. If the return code is SQL_SUCCESS_WITH_INFO or SQL_ERROR, use the CLI diagnostic functions `SQLGetDiagRec()` and `SQLGetDiagField()` to determine why the error occurred.

   If a stored procedure has executed successfully, any variables bound as OUT parameters may contain data that the stored procedure has passed back to the CLI application. If applicable, the stored procedure may also return one or more result sets through non-scrollable cursors. CLI applications should process stored procedure result sets as they would process result sets generated by executing SELECT statements.

   Note: If a CLI application is unsure of the number or type of parameters in a result set returned by a stored procedure, the `SQLNumResultCols()`, `SQLDescribeCol()`, and `SQLColAttribute()` functions can be called (in this order) on the result set to determine this information.

Once you have executed the CALL statement, you can retrieve result sets from the stored procedure if applicable.

Note:

The CLI/ODBC driver will normally autobind the CLI packages the first time a CLI/ODBC application executes SQL against the database, provided the user has the appropriate privilege or authorization. Autobinding of the CLI packages cannot be performed from within a stored procedure, and therefore will not take place if the very first thing an application does is call a CLI stored procedure. Before running a CLI application that calls a CLI stored procedure against a new DB2 database, you must bind the CLI packages once with this command:

UNIX

```
db2 bind <BNDPATH>/@db2cli.lst blocking all
```

Windows

```
db2bind "%DB2PATH%\bnd\@db2cli.lst" blocking
```

The recommended approach is to always bind these packages at the time the database is created to avoid autobind at runtime. Autobind can fail if the user does not have privilege, or if another application tries to autobind at the same time.

Related concepts:

- "DB2 Stored Procedures" in the *Application Development Guide: Programming Client Applications*

Related tasks:

- "Initializing CLI Applications" on page 24
- "Setting Up the CLI Environment" on page 247
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Binding Parameter Markers in CLI Applications" on page 36

Related reference:

- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagField Function (CLI) - Get a Field of Diagnostic Data" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagRec Function (CLI) - Get Multiple Fields Settings of Diagnostic Record" in the *CLI Guide and Reference, Volume 2*
- "CALL statement" in the *SQL Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "CLI Function Return Codes" on page 60

Related samples:

- "spcall.c -- Call individual stored procedures (CLI)"
- "spclient.c -- Call various stored procedures (CLI)"
- "spclires.c -- Contrast stored procedure multiple result set handling methods (CLI)"

- “spserver.c -- Definition of various types of stored procedures (CLI)”

Chapter 9. Compound SQL

Executing SQL Statements in CLI Applications with Compound SQL

Compound SQL allows multiple SQL statements to be grouped into a single executable block. This block of statements, together with any input parameter values, can then be executed in a single continuous stream, reducing the execution time and network traffic.

Restrictions:

- Compound SQL does not guarantee the order in which the substatements are executed, therefore there must not be any dependencies among the substatements.
- Compound SQL statements cannot be nested.
- The BEGIN COMPOUND and END COMPOUND statements must be executed with the same statement handle.
- The value specified in the STOP AFTER FIRST ? STATEMENTS clause of the BEGIN COMPOUND SQL statement must be of type SQL_INTEGER, and you can only bind an application buffer of type SQL_C_INTEGER or SQL_C_SMALLINT for this value.
- Each substatement must have its own statement handle.
- All statement handles must belong to the same connection and have the same isolation level.
- All statement handles must remain allocated until the END COMPOUND statement is executed.
- `SQLEndTran()` cannot be called for the same connection or any connect requests between BEGIN COMPOUND and END COMPOUND.
- Only the following functions may be called using the statement handles allocated for the compound substatements:
  - `SQLAllocHandle()`
  - `SQLBindParameter()`
  - `SQLBindFileToParam()`
  - `SQLExecute()`
  - `SQLExecDirect()`
  - `SQLParamData()`
  - `SQLPrepare()`

– `SQLPutData()`

Procedure:

To execute SQL statements with Compound SQL:

1. Allocate a parent statement handle. For example:
   ```
   SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtparent);
   ```
2. Allocate statement handles for each of the compound substatements. For example:
   ```
   SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub1);
   SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub2);
   SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &hstmtsub3);
   ```
3. Prepare the substatements. For example:
   ```
   SQLPrepare (hstmtsub1, stmt1, SQL_NTS);
   SQLPrepare (hstmtsub2, stmt2, SQL_NTS);
   SQLPrepare (hstmtsub3, stmt3, SQL_NTS);
   ```
4. Execute the BEGIN COMPOUND statement using the parent statement handle. For example:
   ```
   SQLExecDirect (hstmtparent, (SQLCHAR *) "BEGIN COMPOUND NOT ATOMIC STATIC",
                  SQL_NTS);
   ```
5. Execute the substatements. For example:
   ```
   SQLExecute (hstmtsub1);
   SQLExecute (hstmtsub2);
   SQLExecute (hstmtsub3);
   ```
6. Execute the END COMPOUND statement using the parent statement handle. For example:
   ```
   SQLExecDirect (hstmtparent, (SQLCHAR *) "END COMPOUND NOT ATOMIC STATIC",
                  SQL_NTS);
   ```
7. Optional: If you used an input parameter value array, call `SQLRowCount()` with the parent statement handle to retrieve the aggregate number of rows affected by all elements of the input array. For example:
   ```
   SQLRowCount (hstmtparent, &numRows);
   ```
8. Free the handles of the substatements. For example:
   ```
   SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub1);
   SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub2);
   SQLFreeHandle (SQL_HANDLE_STMT, hstmtsub3);
   ```
9. Free the parent statement handle when you have finished using it. For example:
   ```
   SQLFreeHandle (SQL_HANDLE_STMT, hstmtparent);
   ```

If the application is not operating in auto-commit mode and the COMMIT option is not specified, the sub-statements will not be committed. If the application is operating in auto-commit mode, however, then the sub-statements will be committed at END COMPOUND, even if the COMMIT option is not specified.

Related tasks:

- "Allocating Statement Handles in CLI Applications" on page 28
- "Preparing and Executing SQL Statements in CLI Applications" on page 31
- "Freeing Statement Resources in CLI Applications" on page 46

Related reference:

- "SQLAllocHandle Function (CLI) - Allocate Handle" in the *CLI Guide and Reference, Volume 2*
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLExecute Function (CLI) - Execute a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*
- "SQLPrepare Function (CLI) - Prepare a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLRowCount Function (CLI) - Get Row Count" in the *CLI Guide and Reference, Volume 2*
- "COMMIT statement" in the *SQL Reference, Volume 2*
- "ROLLBACK statement" in the *SQL Reference, Volume 2*
- "Compound SQL (Dynamic) statement" in the *SQL Reference, Volume 2*
- "Return Codes for Compound SQL in CLI Applications" on page 149

Related samples:

- "dbuse.c -- How to use a database (CLI)"

Return Codes for Compound SQL in CLI Applications

Return codes are generated on the call to `SQLExecute()` or `SQLExecDirect()` for the END COMPOUND statement. The following lists the return codes for ATOMIC and NOT ATOMIC compound statements:

ATOMIC

- SQL_SUCCESS: all substatements have executed without any warnings or errors.
- SQL_SUCCESS_WITH_INFO: all substatements executed successfully with one or more warnings. Call `SQLError()` to obtain generic diagnostic

information, or call `SQLGetSQLCA()` to obtain the SQLCA for the entire compound SQL statement. The handle used for `SQLError()` or `SQLGetSQLCA()` must be the same one used to process the BEGIN COMPOUND and END COMPOUND statements.

Most of the information in the SQLCA reflects values set by the database server when it processed the last sub-statement, such as the SQLCODE and SQLSTATE. If one or more errors occurred and none of these are of a serious nature, the SQLERRMC field in the SQLCA will contain information on a maximum of seven of these errors.

- SQL_NO_DATA_FOUND: BEGIN COMPOUND and END COMPOUND statements executed without any substatements, or none of the substatements affected any rows.
- SQL_ERROR: one or more substatements failed and all substatements were rolled back.

NOT ATOMIC

- SQL_SUCCESS: all substatements executed without any errors.
- SQL_SUCCESS_WITH_INFO: the COMPOUND statement executed with one or more warnings returned by one or more substatements. Call `SQLError()` or `SQLGetSQLCA()` to retrieve additional information on the warnings.
- SQL_NO_DATA_FOUND: the BEGIN COMPOUND and END COMPOUND statements executed without any substatements, or none of the substatements affected any rows.
- SQL_ERROR: the COMPOUND statement failed. At least one substatement returned an error. Examine the SQLCA to determine which statements failed.

Related tasks:

- "Executing SQL Statements in CLI Applications with Compound SQL" on page 147

Related reference:

- "SQLError Function (CLI) - Retrieve Error Information" in the *CLI Guide and Reference, Volume 2*
- "SQLExecDirect Function (CLI) - Execute a Statement Directly" in the *CLI Guide and Reference, Volume 2*
- "SQLGetSQLCA Function (CLI) - Get SQLCA Data Structure" in the *CLI Guide and Reference, Volume 2*
- "SQLCA (SQL communications area)" in the *SQL Reference, Volume 1*
- "Compound SQL (Dynamic) statement" in the *SQL Reference, Volume 2*

Chapter 10. Multithreaded CLI Applications

Multithreaded CLI Applications

DB2 CLI supports concurrent execution of threads on the following platforms:
- AIX®
- HP-UX
- Linux
- Solaris
- Windows®

On any other platform that supports threads, DB2 CLI is guaranteed to be thread safe by serializing all threaded access to the database. In other words, applications or stored procedures that use DB2 CLI can be invoked multiple times and at the same time.

Note: If you are writing applications that use DB2 CLI calls and either embedded SQL or DB2® API calls, see the documentation for multithreaded mixed applications.

Concurrent execution means that two threads can run independently of each other (on a multi-processor machine they may run simultaneously). For example, an application could implement a database-to-database copy in the following way:
- One thread connects to database A and uses `SQLExecute()` and `SQLFetch()` calls to read data from one connection into a shared application buffer.
- The other thread connects to database B and concurrently reads from the shared buffer and inserts the data into database B.

In contrast, if DB2 CLI serializes all function calls, only one thread may be executing a DB2 CLI function at a time. All other threads would have to wait until the current thread is done before it would get a chance to execute.

When to Use Multiple Threads:

The most common reason to create another thread in a DB2 CLI application is so a thread other than the one executing can be used to call `SQLCancel()` (to cancel a long running query for example).

Most GUI based applications use threads in order to ensure that user interaction can be handled on a higher priority thread than other application tasks. The application can simply delegate one thread to run all DB2 CLI functions (with the exception of `SQLCancel()`). In this case there are no thread-related application design issues since only one thread will be accessing the data buffers that are used to interact with DB2 CLI.

Applications that use multiple connections, and are executing statements that may take some time to execute, should consider executing DB2 CLI functions on multiple threads to improve throughput. Such an application should follow standard practices for writing any multi-threaded application, most notably, those concerned with sharing data buffers.

Programming Tips:

Any resource allocated by DB2 CLI is guaranteed to be thread-safe. This is accomplished by using either a shared global or connection specific semaphore. At any one time, only one thread can be executing a DB2 CLI function that accepts an environment handle as input. All other functions that accept a connection handle (or a statement or descriptor allocated on that connection handle) will be serialized on the connection handle.

This means that once a thread starts executing a function with a connection handle, or child of a connection handle, any other thread will block and wait for the executing thread to return. The one exception to this is `SQLCancel()`, which must be able to cancel a statement currently executing on another thread. For this reason, the most natural design is to map one thread per connection, plus one thread to handle `SQLCancel()` requests. Each thread can then execute independently of the others.

If an object is shared across threads, application timing issues may arise. For example, if a thread is using a handle in one thread, and another thread frees that handle between function calls, the next attempt to use that handle would result in a return code of SQL_INVALID_HANDLE.

Notes:

1. Thread safety for handles only applies for DB2 CLI applications. ODBC applications may trap since the handle in this case is a pointer and the pointer may no longer be valid if another thread has freed it. For this reason, it is best when writing an ODBC application to follow the application model for multithreaded CLI applications.
2. There may be platform or compiler specific link options required for multi-threaded applications. Refer to your compiler documentation for further details.

Related concepts:

- “Application Model for Multithreaded CLI Applications” on page 153
- “Mixed Multithreaded CLI Applications” on page 154

Related reference:

- “SQLCancel Function (CLI) - Cancel Statement” in the *CLI Guide and Reference, Volume 2*
- “CLI Function Return Codes” on page 60

Application Model for Multithreaded CLI Applications

The following model of a typical multithreaded CLI application is intended as an example:

- Designate a master thread which allocates:
  - *m* "child" threads
  - *n* connection handles
- Each task that requires a connection is executed by one of the child threads, and is given one of the *n* connections by the master thread.
- Each connection is marked as in use by the master thread until the child thread returns it to the master thread.
- Any `SQLCancel()` request is handled by the master thread.

This model allows the master thread to have more threads than connections if the threads are also used to perform non-SQL related tasks, or more connections than threads if the application wants to maintain a pool of active connections to various databases, but limit the number of active tasks.

Note: A multithreaded DB2 CLI stored procedure can only connect to the database where the stored procedure is currently executing.

Most importantly, this ensures that two threads are not trying to use the same connection handle at any one time. Although DB2 CLI controls access to its resources, the application resources such as bound columns and parameter buffers are not controlled by DB2 CLI, and the application must guarantee that a pointer to a buffer is not being used by two threads at any one time. Any deferred arguments must remain valid until the column or parameter has been unbound.

If it is necessary for two threads to share a data buffer, the application must implement some form of synchronization mechanism. For example, in the database-to-database copy scenario where one thread connects to database A and reads data from one connection into a shared application buffer while the other thread connects to database B and concurrently reads from the shared buffer and inserts data into database B, the use of the shared buffer must be synchronized by the application.

Application Deadlocks:

The application must be aware of the possibility of creating deadlock situations with shared resources in the database and the application.

DB2® can detect deadlocks at the server and rollback one or more transactions to resolve them. An application may still deadlock if:
- two threads are connected to the same database, and
- one thread is holding an application resource ’A’ and is waiting for a database resource ’B’, and
- the other thread has a lock on the database resource ’B’ while waiting for the application resource ’A’.

In this case the DB2 server is only going to see a lock, not a deadlock, and unless the database LOCKTIMEOUT configuration setting is set, the application will wait forever.

The model suggested above avoids this problem by not sharing application resources between threads once a thread starts executing on a connection.

Related concepts:
- “Multithreaded CLI Applications” on page 151
- “Mixed Multithreaded CLI Applications” on page 154

Related reference:
- “Lock Timeout configuration parameter - locktimeout” in the *Administration Guide: Performance*
- “SQLCancel Function (CLI) - Cancel Statement” in the *CLI Guide and Reference, Volume 2*

Mixed Multithreaded CLI Applications

It is possible for a multi-threaded application to mix CLI calls with DB2® API calls and embedded SQL. The type of the call executed earliest in the application determines the best way to organize the application:

DB2 CLI Calls First

The DB2 CLI driver automatically calls the DB2 context APIs to allocate and manage contexts for the application. This means that any application that calls `SQLAllocEnv()` before calling any other DB2 API or embedded SQL will be initialized with the context type set to SQL_CTX_MULTI_MANUAL.

In this case the application should allow DB2 CLI to allocate and manage all contexts. Use DB2 CLI to allocate all connection handles and to perform all connections. Call the `SQLSetConnect()` function in each thread prior to calling any embedded SQL. DB2 APIs can be called after any DB2 CLI function has been called in the same thread.

DB2 API or Embedded SQL Calls First

The DB2 CLI driver does not automatically call the DB2 context APIs if the application calls any DB2 API or embedded SQL functions before a CLI function.

This means that any thread that calls a DB2 API or embedded SQL function must be attached to a context, otherwise the call will fail with an SQLCODE of SQL1445N. This can be done by calling the DB2 API `sqleAttachToCtx()` which will explicitly attach the thread to a context, or by calling any DB2 CLI function (`SQLSetConnection()` for example). In this case, the application must explicitly manage all contexts.

Use the context APIs to allocate and attach to contexts prior to calling DB2 CLI functions (`SQLAllocEnv()` will use the existing context as the default context). Use the SQL_ATTR_CONN_CONTEXT connection attribute to explicitly set the context that each DB2 CLI connection should use.

Note: It is recommended that you do not use the default application stack size, but instead increase the stack size to at least 256 000. DB2 requires a minimum application stack size of 256 000 when calling a DB2 function. You must ensure therefore, that you allocate a total stack size that is large enough for both your application and the minimum requirements for a DB2 function call.

Related concepts:
- "DB2 CLI versus Embedded Dynamic SQL" in the *Application Development Guide: Programming Client Applications*
- "Multithreaded CLI Applications" on page 151
- "Application Model for Multithreaded CLI Applications" on page 153

Related reference:
- "SQLAllocEnv Function (CLI) - Allocate Environment Handle" in the *CLI Guide and Reference, Volume 2*
- "sqleAttachToCtx - Attach to Context" in the *Administrative API Reference*
- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

Chapter 11. Multisite Updates (Two Phase Commit)

Multisite Updates (Two Phase Commit) in CLI Applications

A typical transaction scenario portrays an application which interacts with only one database server in a transaction. Even though concurrent connections allow for concurrent transactions, the different transactions are not coordinated.

With multisite updates, the two phase commit (2PC) protocol, and coordinated distributed transactions, an application is able to update data in multiple remote database servers with guaranteed integrity.

Note: Multisite update is also known as Distributed Unit of Work (DUOW).

A typical banking transaction is a good example of a multisite update. Consider the transfer of money from one account to another in a different database server. In such a transaction it is critical that the updates that implement the debit operation on one account do not get committed unless the updates required to process the credit to the other account are committed as well. Multisite update considerations apply when data representing these accounts is managed by two different database servers

Some multisite updates involve the use of a transaction manager (TM) to coordinate two-phase commit among multiple databases. DB2 CLI applications can be written to use various transaction managers:
- DB2® as transaction manager
- Process-based XA-compliant transaction program monitor
- Host and AS/400® database servers

  Note: There is no specific DB2 CLI/ODBC client configuration required when connecting to a host or iSeries™ database server, although the machine running DB2 Connect™ may require certain configuration settings to enable running multisite update mode against the host.

Related concepts:
- "Multisite Updates" in the *DB2 Connect User's Guide*

- “Configuration Parameter Considerations for Multisite Update Applications” in the *Application Development Guide: Programming Client Applications*
- “DB2 as Transaction Manager in CLI Applications” on page 158
- “Process-based XA-Compliant Transaction Program Monitor (XA TP) Programming Considerations for CLI Applications” on page 164

Related tasks:

- “Enabling Multisite Updates using the Control Center” in the *DB2 Connect User’s Guide*

DB2 as Transaction Manager in CLI Applications

Configuration of DB2 as Transaction Manager:

DB2 CLI/ODBC applications can use DB2® itself as the Transaction Manager (DB2 TM) to coordinate distributed transactions against all IBM® database servers.

The DB2 Transaction Manager must be set up according to the information in the DB2 transaction manager configuration documentation.

To use DB2 as the transaction manager in CLI/ODBC applications, the following CLI/ODBC configuration keywords must be set in the [COMMON] section of the db2cli.ini file as follows:

```
[COMMON]
DISABLEMULTITHREAD = 1
CONNECTTYPE=2
SYNCPOINT=2
```

Two of the above configuration keywords can also be set using the following environment attributes:

- SQL_ATTR_CONNECT_TYPE to SQL_COORDINATED_TRANS
- SQL_ATTR_SYNCPOINT to SQL_TWOPHASE

There is no appropriate environment attribute for the DISABLEMULTITHREAD keyword which must therefore still be set to 1 in the [COMMON] section of the db2cli.ini file.

Because the DISABLEMULTITHREAD keyword must appear in the [COMMON] section of the db2cli.ini file it impacts all connections to all data sources from that client instance. This means that a DB2 client instance can only support process-based CLI applications or thread-based CLI applications, but not both.

Programming Considerations:

You must set the DB2 CLI/ODBC configuration keyword DISABLEMULTITHREAD to 1. This indicates that the DB2 CLI/ODBC driver will use a single DB2 context for all connections made by the application process. All database requests will be serialized at the process level.

The environment attribute SQL_ATTR_CONNECTTYPE controls whether the application is to operate in a coordinated or uncoordinated distributed environment. Commits or rollbacks among multiple database connections are coordinated in a coordinated distributed environment. The two possible values for this attribute are:

- SQL_CONCURRENT_TRANS - supports single database per transaction semantics. Multiple concurrent connections to the same database and to different databases are permitted. Each connection has its own commit scope. No effort is made to enforce coordination of transactions. This is the default and corresponds to a Type 1 CONNECT in embedded SQL. The current setting of the SQL_ATTR_SYNC_POINT option is ignored.
- SQL_COORDINATED_TRANS - supports the multiple databases per transaction semantics, as discussed below. Setting SQL_ATTR_CONNECTTYPE to this value corresponds to Type 2 CONNECT in embedded SQL.

All connections within an application must have the same SQL_ATTR_CONNECTTYPE setting. It is recommended that the application set this environment attribute with a call to `SQLSetEnvAttr()`, if necessary, as soon as the environment handle has been allocated. Since ODBC applications cannot access `SQLSetEnvAttr()`, they must set this using `SQLSetConnectAttr()` after each connection handle is allocated, but before any connections have been established.

Attributes that Govern Multisite Update Semantics:

A coordinated transaction is one in which commits or rollbacks among multiple database connections are coordinated. The SQL_COORDINATED_TRANS setting of the SQL_ATTR_CONNECTTYPE attribute corresponds to the Type 2 CONNECT in IBM embedded SQL and must be considered in conjunction with the SQL_ATTR_SYNC_POINT attribute, which has the following two possible settings:

- SQL_ONEPHASE: One-phase commit is used to commit the work done by each database in a multiple database transaction. To ensure data integrity, each transaction must not update more than one database. The first database that has updates performed in a transaction becomes the only updater in that transaction; all other databases accessed are treated as read-only. Any update that is attempted to a read-only database within the transaction will result in the entire transaction being rolled back. (This is a change in behavior from previous versions of DB2.) An SQLCODE will be returned to the application indicating this.

- SQL_TWOPHASE: Two-phase commit is used to commit the work done by each database in a multiple database transaction. This requires the use of a Transaction Manager to coordinate two phase commits amongst the databases that support this protocol. Multiple readers and multiple updaters are allowed within a transaction.

The application should set this environment attribute as soon as the environment handle has been created. It should be set with a call to `SQLSetEnvAttr()`. ODBC applications must use `SQLSetConnectAttr()` to set this for each connection handle under the environment before any connections have been established.

All the connections within an application must have the same SQL_ATTR_CONNECTTYPE and SQL_ATTR_SYNC_POINT settings. After the first connection has been established, all subsequent connect types must be the same as the first. Coordinated connections default to manual-commit mode.

The function `SQLEndTran()` must be used in a multisite update environment when DB2 is acting as the transaction manager.

Figure 10 on page 161 shows the logical flow of an application executing statements on two SQL_CONCURRENT_TRANS connections ('A' and 'B'), and indicates the scope of the transactions.

Figure 11 on page 162 shows the same statements being executed on two SQL_COORDINATED_TRANS connections ('A' and 'B'), and the scope of a coordinated distributed transaction.

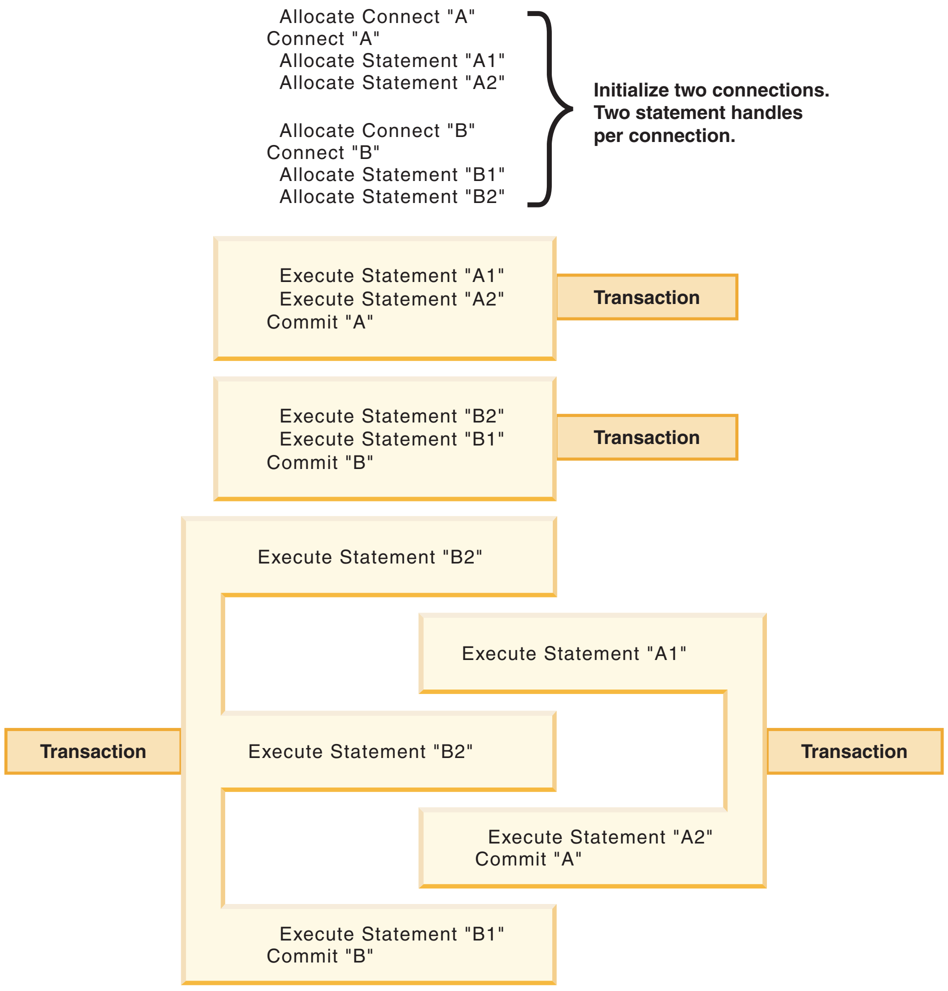


Figure 10. Multiple Connections with Concurrent Transactions

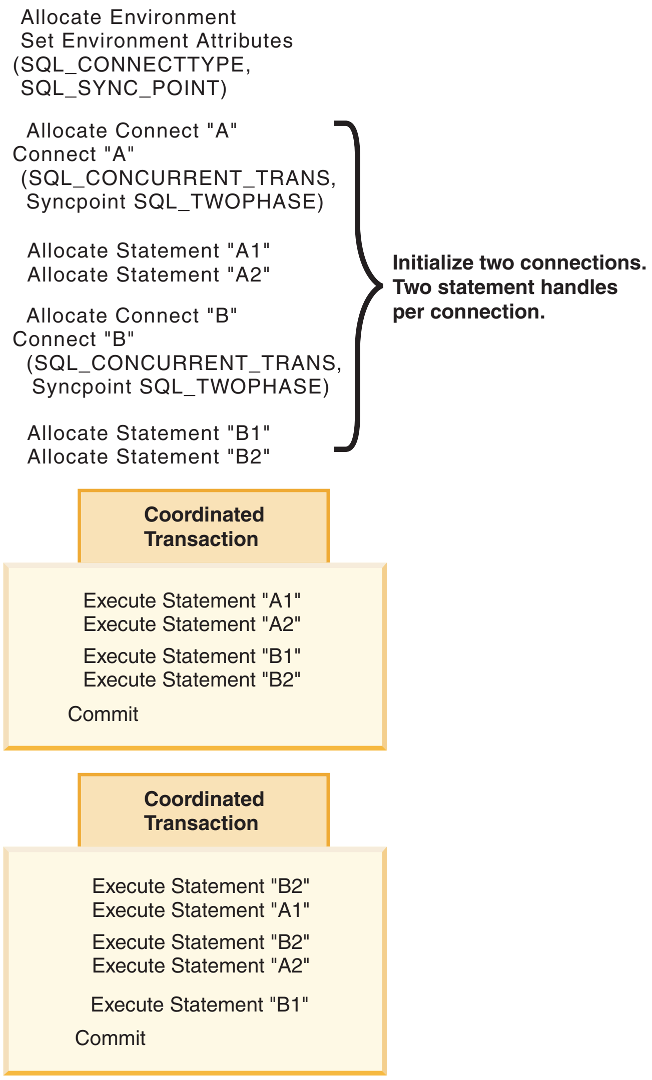


Figure 11. Multiple Connections with Coordinated Transactions

Establishing a Coordinated Transaction Connection:

An application can establish coordinated transaction connections by calling the `SQLSetEnvAttr()` function, or by setting the CONNECTTYPE and SYNCPOINT keywords in the DB2 CLI initialization file or in the connection string for `SQLDriverConnect()`. The initialization file is intended for existing applications that cannot use the `SQLSetEnvAttr()` or `SQLSetConnectAttr()` functions.

An application cannot have a mixture of concurrent and coordinated connections; the type of the first connection will determine the type of all subsequent connections. `SQLSetEnvAttr()` will return an error if an application attempts to change the connect type while there is an active connection.

Restrictions:

Mixing embedded SQL and CLI/ODBC calls in a multisite update environment is supported, but all the same restrictions of writing mixed applications are imposed.

Related concepts:

- "Considerations for Mixing Embedded SQL and DB2 CLI" on page 215
- "DB2 transaction manager configuration" in the *Administration Guide: Planning*
- "Multisite Updates (Two Phase Commit) in CLI Applications" on page 157
- "db2cli.ini Initialization File" on page 293

Related reference:

- "SQLEndTran Function (CLI) - End Transactions of a Connection or an Environment" in the *CLI Guide and Reference, Volume 2*
- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*
- "SQLSetEnvAttr Function (CLI) - Set Environment Attribute" in the *CLI Guide and Reference, Volume 2*
- "Environment Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CONNECTYPE CLI/ODBC Configuration Keyword" on page 306
- "DISABLEMULTITHREAD CLI/ODBC Configuration Keyword" on page 317
- "SYNCPOINT CLI/ODBC Configuration Keyword" on page 336

Related samples:

- "dbmcon.c -- How to use multiple databases (CLI)"
- "dbmconx.c -- How to use multiple databases with embedded SQL. (CLI)"

Process-based XA-Compliant Transaction Program Monitor (XA TP) Programming Considerations for CLI Applications

Process-based XA TPs, such as CICS® and Encina, start up one application server per process. In each application-server process, the connections are already established using the XA API (xa_open). This section describes the environment configurations and considerations for running DB2 CLI/ODBC applications under this environment.

Configuration:

The XA Transaction Manager must be set up according to the configuration considerations for XA transaction managers.

Note: Setting the CLI/ODBC configuration keywords for connections is no longer required when in an XA Transactional processing environment.

Programming Considerations:

DB2 CLI/ODBC applications written for this environment must complete the following steps:

- The application must first call `SQLConnect()` or `SQLDriverConnect()` to associate the TM-opened connections with the CLI/ODBC connection handle. The data source name must be specified. User ID and Password are optional.
- The application must call the XA TM to do a commit or rollback. As a result, since the CLI/ODBC driver does not know that the transaction has ended, the application should do the following before exiting:
  - Drop all CLI/ODBC statement handles.
  - Free up the connection handle by calling `SQLDisconnect()` and `SQLFreeHandle()`. The actual database connection will not be disconnected until the XA TM performs an xa_close.

Restrictions:

Mixing embedded SQL and CLI/ODBC calls in a multisite update environment is supported, but all the same restrictions of writing mixed applications are imposed.

Related concepts:

- "Considerations for Mixing Embedded SQL and DB2 CLI" on page 215
- "Configuration considerations for XA transaction managers" in the *Administration Guide: Planning*
- "DB2 as Transaction Manager in CLI Applications" on page 158

- “Multisite Updates (Two Phase Commit) in CLI Applications” on page 157

Related reference:

- “SQLConnect Function (CLI) - Connect to a Data Source” in the *CLI Guide and Reference, Volume 2*
- “SQLDisconnect Function (CLI) - Disconnect from a Data Source” in the *CLI Guide and Reference, Volume 2*
- “SQLDriverConnect Function (CLI) - (Expanded) Connect to a Data Source” in the *CLI Guide and Reference, Volume 2*
- “SQLFreeHandle Function (CLI) - Free Handle Resources” in the *CLI Guide and Reference, Volume 2*

Chapter 12. Unicode

Unicode CLI Applications

There are two main areas of support for DB2® CLI Unicode applications:

- The addition of a set of functions that accept Unicode string arguments in place of ANSI string arguments.
- The addition of new C and SQL data types to describe Unicode data.

To be considered a Unicode application, the application must set the SQL_ATTR_ANSI_APP connection attribute to SQL_AA_FALSE, before a connection is made. This will ensure that CLI will consider Unicode the preferred method of communication between itself and the database.

ODBC adds types to the set of C and SQL types that already exist to accommodate Unicode, and CLI uses these additional types accordingly. The new C type, SQL_C_WCHAR, indicates that the C buffer contains UCS-2 data in native-endian format. The new SQL types, SQL_WCHAR, SQL_WVARCHAR, and SQL_WLONGVARCHAR, indicate that a particular column or parameter marker contains Unicode data. For DB2 Unicode databases, graphic columns will be described using the new types. Conversion will be allowed between SQL_C_WCHAR and SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR and SQL_CLOB, as well as with the graphic data types.

Obsolete CLI/ODBC Keyword Values:

Before Unicode applications were supported, applications that were written to work with single-byte character data could be made to work with double-byte graphic data by a series of DB2 CLI configuration keywords, such as GRAPHIC=1,2 or 3, Patch2=7. These workarounds presented graphic data as character data, and also affected the reported length of the data. These keywords are no longer required for Unicode applications, and should not be used due to the risk of potential side effects. If it is not known if a particular application is a Unicode application, we suggest you try without any of the keywords that affect the handling of graphic data.

Literals in Unicode Databases:

In non-Unicode databases, data in LONG VARGRAPHIC and LONG VARCHAR columns cannot be compared. Data in GRAPHIC/VARGRAPHIC and CHAR/VARCHAR columns can only be compared, or assigned to each other, using explicit cast functions since no implicit code page conversion is supported. This includes GRAPHIC/VARGRAPHIC and CHAR/VARCHAR literals where a GRAPHIC/VARGRAPHIC literal is differentiated from a CHAR/VARCHAR literal by a G prefix. For Unicode databases, casting between GRAPHIC/VARGRAPHIC and CHAR/VARCHAR literals is not required. Also, a G prefix is not required in front of a GRAPHIC/VARGRAPHIC literal. Provided at least one of the arguments is a literal, implicit conversions occur. This allows literals with or without the G prefix to be used within statements that use either `SQLPrepareW()` or `SQLExecDirect()`. Literals for LONG VARGRAPHICs still must have a G prefix.

Related concepts:

- "Unicode Functions (CLI)" on page 168
- "ODBC Unicode Applications" on page 169

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "PATCH2 CLI/ODBC Configuration Keyword" on page 326

Unicode Functions (CLI)

DB2 CLI Unicode functions accept Unicode string arguments in place of ANSI string arguments. ODBC API functions have suffixes to indicate the format of their string arguments: those that accept Unicode end in W, and those that accept ANSI have no suffix (ODBC adds equivalent functions with names that end in A, but these are not offered by DB2 CLI). The following is a list of functions available in DB2 CLI which have both ANSI and Unicode versions:

```
SQLBrowseConnect        SQLForeignKeys          SQLPrimaryKeys
SQLColAttribute         SQLGetConnectAttr       SQLProcedureColumns
SQLColAttributes        SQLGetConnectOption     SQLProcedures
SQLColumnPrivileges     SQLGetCursorName        SQLSetConnectAttr
SQLColumns              SQLGetDescField         SQLSetConnectOption
SQLConnect              SQLGetDescRec           SQLSetCursorName
SQLDataSources          SQLGetDiagField         SQLSetDescField
SQLDescribeCol          SQLGetDiagRec           SQLSetStmtAttr
SQLDriverConnect        SQLGetInfo              SQLSpecialColumns
SQLError                SQLGetStmtAttr          SQLStatistics
SQLExecDirect           SQLNativeSQL            SQLTablePrivileges
SQLExtendedPrepare      SQLPrepare              SQLTables
```

Unicode functions that have arguments which are always the length of strings interpret these arguments as count-of-characters. For functions that return length information for server data, the display size and precision are again described in terms of characters. When the length (transfer size of the data) could refer to string or nonstring data, it is interpreted as a count-of-bytes. For example, `SQLGetInfoW()` will still take the length as count-of-bytes, but `SQLExecDirectW()` will use count-of-characters. CLI will return data from result sets in either Unicode or ANSI, depending on the application's binding. If an application binds to SQL_C_CHAR, the driver will convert SQL_WCHAR data to SQL_CHAR. An ODBC Driver Manager, if used, maps SQL_C_WCHAR to SQL_C_CHAR for ANSI drivers but does no mapping for Unicode drivers.

ANSI to Unicode Function Mappings:

The syntax for a DB2 CLI Unicode function is the same as the syntax for its corresponding ANSI function, except that SQLCHAR parameters are defined as SQLWCHAR. Character buffers defined as SQLPOINTER in the ANSI syntax can be defined as either SQLCHAR or SQLWCHAR in the Unicode function. Refer to the ANSI version of the CLI Unicode functions for ANSI syntax details.

Related concepts:
- "Unicode CLI Applications" on page 167
- "ODBC Unicode Applications" on page 169

Related reference:
- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "CLI and ODBC Function Summary" in the *CLI Guide and Reference, Volume 2*

ODBC Unicode Applications

A Unicode ODBC application sends and retrieves character data primarily in UCS-2. It does this by calling Unicode versions of the ODBC functions (those with a 'W' suffix) and by indicating Unicode data types. The application does not explicitly specify a local code page. The application can still call the ANSI functions and pass local code page strings.

For example, the application may call SQLConnectW() and pass the DSN, User ID and Password as Unicode arguments. It may then call SQLExecDirectW() and pass in a Unicode SQL statement string, and then bind a combination of ANSI local code page buffers (SQL_C_CHAR) and Unicode buffers (SQL_C_WCHAR). The database data types may or may not be Unicode.

If a CLI application calls `SQLSetConnectAttr()` with SQL_ATTR_ANSI_APP set to SQL_AA_FALSE or calls SQLConnectW without setting the value of SQL_ATTR_ANSI_APP, then the application is considered a Unicode application. This means all CHAR data is sent and received from a Unicode database in UTF-8 format. The application can then fetch CHAR data into SQL_C_CHAR buffers in local code page (with possible data loss), or into SQL_C_WCHAR buffers in UCS-2 without any data loss.

If the application does not do either of the two calls above, CHAR data is converted to the applications local code page at the server. This means CHAR data fetched into SQL_C_WCHAR may suffer data loss.

If the DB2CODEPAGE instance variable is set (using db2set) to code page 1208 (UTF-8), the application will receive all CHAR data in UTF-8 since this is now the local code page. The application must also ensure that all CHAR input data is also in UTF-8. ODBC also assumes that all SQL_C_WCHAR data is in the native endian format. CLI will perform any required byte-reversal for SQL_C_WCHAR.

ODBC Unicode Versus Non-Unicode Applications:

A Unicode driver must export `SQLConnectW()` in order to be recognized as a Unicode driver by the driver manager. It is important to note that many ODBC applications (such as Microsoft® Access and Visual Basic) call `SQLConnectW()`. In previous releases of DB2® Universal Database, DB2 CLI has not supported this API, and thus was not recognized as a Unicode driver by the ODBC driver manager. This caused the ODBC driver manager to convert all Unicode data to the application's local code page. With the added support of the `SQLConnectW()` function, these applications will now connect as Unicode applications and DB2 CLI will take care of all required data conversion.

DB2 CLI accepts Unicode APIs (with a suffix of ″W″) and regular ANSI APIs. ODBC defines a set of functions with a suffix of ″A″, but the driver manager does not pass ANSI functions with the ″A″ suffix to the driver. Instead, it converts these functions to ANSI function calls without the suffix, and then passes them to the driver.

An ODBC application that calls the `SQLConnectW()` API is considered a Unicode application. Since the ODBC driver manager will always call the `SQLConnectW()` API regardless of what version the application called, ODBC introduced the SQL_ATTR_ANSI_APP connect attribute to notify the driver if the application should be considered an ANSI or UNICODE application. If

SQL_ATTR_ANSI_APP is set to SQL_AA_TRUE, the DB2 CLI converts all Unicode data to the local code page before sending it to the server.

Related concepts:

- "Unicode character encoding" in the *Administration Guide: Planning*
- "Unicode Functions (CLI)" on page 168
- "Unicode CLI Applications" on page 167

Related reference:

- "C Data Types for CLI Applications" on page 54
- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*

Chapter 13. User Defined Types (UDT)

Distinct Type Usage in CLI Applications

In addition to SQL data types (referred to as *base* SQL data types), new distinct types can be defined by the user. This variety of user defined types (UDTs) shares its internal representation with an existing type, but is considered to be a separate and incompatible type for most operations. Distinct types are created using the CREATE DISTINCT TYPE SQL statement.

Distinct types help provide the strong typing control needed in object oriented programming by ensuring that only those functions and operators explicitly defined on a distinct type can be applied to its instances. Applications continue to work with C data types for application variables, and only need to consider the distinct types when constructing SQL statements.

This means:
- All SQL to C data type conversion rules that apply to the built-in type apply to distinct types.
- Distinct types will have the same default C Type as the built-in type.
- `SQLDescribeCol()` will return the built-in type information. The user defined type name can be obtained by calling `SQLColAttribute()` with the input descriptor type set to SQL_DESC_DISTINCT_TYPE.
- SQL predicates that involve parameter markers must be explicitly cast to the distinct type. This is required since the application can only deal with the built-in types, so before any operation can be performed using the parameter, it must be cast from the C built-in type to the distinct type; otherwise an error will occur when the statement is prepared.

Related concepts:
- "User Defined Type (UDT) Usage in CLI Applications" on page 174

Related reference:
- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "SQLColAttribute Function (CLI) - Return a Column Attribute" in the *CLI Guide and Reference, Volume 2*
- "SQLDescribeCol Function (CLI) - Return a Set of Attributes for a Column" in the *CLI Guide and Reference, Volume 2*
- "CREATE DISTINCT TYPE statement" in the *SQL Reference, Volume 2*

- "SQL to C Data Conversion in CLI" on page 360

Related samples:

- "dtudt.c -- How to create, use, and drop user-defined distinct types. (CLI)"

User Defined Type (UDT) Usage in CLI Applications

User Defined Types (UDTs) are database types defined by the user to provide structure or strong typing not available with conventional SQL types. There are three varieties of UDT: distinct types, structured types, and reference types.

A CLI application may want to determine whether a given database column is a UDT, and if so, the variety of UDT. The descriptor field SQL_DESC_USER_DEFINED_TYPE_CODE may be used to obtain this information. When SQL_DESC_USER_DEFINED_TYPE_CODE is retrieved using `SQLColAttribute()` or directly from the IPD using `SQLGetDescField()`, it will have one of the following numeric values:

```
SQL_TYPE_BASE  (this is a regular SQL type, not a UDT)
SQL_TYPE_DISTINCT  (this value indicates that the column
                          is a distinct type)
SQL_TYPE_STRUCTURED  (this value indicates that the column
                            is a structured type)
SQL_TYPE_REFERENCE  (this value indicates that the column
                           is a reference type)
```

Additionally, the following descriptor fields may be used to obtain the type names:

- SQL_DESC_REFERENCE_TYPE contains the name of the reference type, or an empty string if the column is not a reference type.
- SQL_DESC_STRUCTURED_TYPE contains the name of the structured type, or an empty string if the column is not a structured type.
- SQL_DESC_USER_TYPE or SQL_DESC_DISTINCT_TYPE contains the name of the distinct type, or an empty string if the column is not a distinct type.

The descriptor fields listed above return the schema as part of the name. If the schema is less than 8 letters, it is padded with blanks.

The connection attribute SQL_ATTR_TRANSFORM_GROUP allows an application to set the transform group, and is an alternative to the SQL statement SET CURRENT DEFAULT TRANSFORM GROUP.

A CLI application may not wish to repeatedly obtain the value of the SQL_DESC_USER_DEFINED_TYPE_CODE descriptor field to determine if columns contain UDTs. For this reason, there is an attribute called SQL_ATTR_RETURN_USER_DEFINED_TYPES at both the connection and the statement handle level. When set to SQL_TRUE using SQLSetConnectAttr(), CLI returns SQL_DESC_USER_DEFINED_TYPE where you would normally find SQL types in results from calls to SQLColAttribute(), SQLDescribeCol() and SQLGetDescField(). This allows the application to check for this special type, and then do special processing for UDTs. The default value for this attribute is SQL_FALSE.

When the SQL_ATTR_RETURN_USER_DEFINED_TYPES attribute is set to SQL_TRUE, the descriptor field SQL_DESC_TYPE will no longer return the ″base″ SQL type of the UDT, that is, the SQL type that the UDT is based on or transforms to. For this reason, the descriptor field SQL_DESC_BASE_TYPE will always return the base type of UDTs, and the SQL type of normal columns. This field simplifies modules of a program that do not deal specifically with UDTs that would otherwise have to change the connection attribute.

Note that SQLBindParameter() will not allow you to bind a parameter of the type SQL_USER_DEFINED_TYPE. You must still bind parameters using the base SQL type, which you can obtain using the descriptor field SQL_DESC_BASE_TYPE. For example, here is the SQLBindParameter() call used when binding to a column with a distinct type based on SQL_VARCHAR:

```
sqlrc = SQLBindParameter (hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
            SQL_VARCHAR, 30, 0, &c2, 30, NULL);
```

Related concepts:

- “Distinct Type Usage in CLI Applications” on page 173

Related reference:

- “SQL Symbolic and Default Data Types for CLI Applications” on page 51
- “SQLColAttribute Function (CLI) - Return a Column Attribute” in the *CLI Guide and Reference, Volume 2*
- “SQLGetDescField Function (CLI) - Get Single Field Settings of Descriptor Record” in the *CLI Guide and Reference, Volume 2*
- “SQLSetConnectAttr Function (CLI) - Set Connection Attributes” in the *CLI Guide and Reference, Volume 2*
- “CREATE DISTINCT TYPE statement” in the *SQL Reference, Volume 2*
- “SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator” in the *CLI Guide and Reference, Volume 2*

Related samples:

- “dtudt.c -- How to create, use, and drop user-defined distinct types. (CLI)”
- “udfcli.c -- How to work with different types of user-defined functions (UDFs) (CLI)”

Chapter 14. Descriptors

Descriptors in CLI Applications

DB2 CLI stores information (data types, size, pointers, and so on) about columns in a result set, and parameters in an SQL statement. The bindings of application buffers to columns and parameters must also be stored. *Descriptors* are a logical view of this information, and provide a way for applications to query and update this information.

Many CLI functions make use of descriptors, but the application itself does not need to manipulate them directly.

For instance:

- When an application binds column data using `SQLBindCol()`, descriptor fields are set that completely describe the binding.
- A number of statement attributes correspond to the header fields of a descriptor. In this case you can achieve the same effect calling `SQLSetStmtAttr()` as calling the corresponding function `SQLSetDescField()` that sets the values in the descriptor directly.

Although no database operations require direct access to descriptors, there are situations where working directly with the descriptors will be more efficient or result in simpler code. For instance, a descriptor that describes a row fetched from a table can then be used to describe a row inserted back into the table.

There are four types of descriptors:

Application Parameter Descriptor (APD)
Describes the application buffers (pointers, data types, scale, precision, length, maximum buffer length, and so on) that are bound to parameters in an SQL statement. If the parameters are part of a CALL statement they may be input, output, or both. This information is described using the application’s C data types.

Application Row Descriptor (ARD)
Describes the application buffers bound to the columns. The application may specify different data types from those in the

implementation row descriptor (IRD) to achieve data conversion of column data. This descriptor reflects any data conversion that the application may specify.

Implementation Parameter Descriptor (IPD)
Describes the parameters in the SQL statement (SQL type, size, precision, and so on).
- If the parameter is used as input, this describes the SQL data that the database server will receive after DB2 CLI has performed any required conversion.
- If the parameter is used as output, this describes the SQL data before DB2 CLI performs any required conversion to the application's C data types.

Implementation Row Descriptor (IRD)
Describes the row of data from the result set before DB2 CLI performs any required data conversion to the application's C data types.

The only difference between the four types of descriptors described above is how they are used. One of the benefits of descriptors is that a single descriptor can be used to serve multiple purposes. For instance, a row descriptor in one statement can be used as a parameter descriptor in another statement.

As soon as a descriptor exists, it is either an application descriptor or an implementation descriptor. This is the case even if the descriptor has not yet been used in a database operation. If the descriptor is allocated by the application using `SQLAllocHandle()` then it is an application descriptor.

Values Stored in a Descriptor:

Each descriptor contains both header fields and record fields. These fields together completely describe the column or parameter.

Header Fields:

Each header field occurs once in each descriptor. Changing one of these fields affects all columns or parameters.

Many of the following header fields correspond to a statement attribute. Setting the header field of the descriptor using `SQLSetDescField()` is the same as setting the corresponding statement attribute using `SQLSetStmtAttr()`. The same holds true for retrieving the information using `SQLGetDescField()` or `SQLGetStmtAttr()`. If your application does not already have a descriptor handle allocated then it is more efficient to use the statement attribute calls instead of allocating the descriptor handle, and then using the descriptor calls.

Table 10. Header fields

SQL_DESC_ALLOC_TYPE	SQL_DESC_BIND_TYPE[a]
SQL_DESC_ARRAY_SIZE[a]	SQL_DESC_COUNT
SQL_DESC_ARRAY_STATUS_PTR[a]	SQL_DESC_ROWS_PROCESSED_PTR[a]
SQL_DESC_BIND_OFFSET_PTR[a]	

Note:

a This header field corresponds to a statement attribute.

The descriptor header field SQL_DESC_COUNT is the one-based index of the highest-numbered descriptor record that contains information (and not a count of the number of columns or parameters). DB2 CLI automatically updates this field (and the physical size of the descriptor) as columns or parameters are bound and unbound. The initial value of SQL_DESC_COUNT is 0 when a descriptor is first allocated.

Descriptor Records:

Zero or more descriptor records are contained in a single descriptor. As new columns or parameters are bound, new descriptor records are added to the descriptor. When a column or parameter is unbound, the descriptor record is removed.

Table 11 lists the fields in a descriptor record. They describe a column or parameter, and occur once in each descriptor record.

Table 11. Record Fields

SQL_DESC_AUTO_UNIQUE_VALUE	SQL_DESC_LOCAL_TYPE_NAME
SQL_DESC_BASE_COLUMN_NAME	SQL_DESC_NAME
SQL_DESC_BASE_TABLE_NAME	SQL_DESC_NULLABLE
SQL_DESC_CASE_SENSITIVE	SQL_DESC_OCTET_LENGTH
SQL_DESC_CATALOG_NAME	SQL_DESC_OCTET_LENGTH_PTR
SQL_DESC_CONCISE_TYPE	SQL_DESC_PARAMETER_TYPE
SQL_DESC_DATA_PTR	SQL_DESC_PRECISION
SQL_DESC_DATETIME_INTERVAL_CODE	SQL_DESC_SCALE
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQL_DESC_SCHEMA_NAME
SQL_DESC_DISPLAY_SIZE	SQL_DESC_SEARCHABLE
SQL_DESC_FIXED_PREC_SCALE	SQL_DESC_TABLE_NAME
SQL_DESC_INDICATOR_PTR	SQL_DESC_TYPE
SQL_DESC_LABEL	SQL_DESC_TYPE_NAME
SQL_DESC_LENGTH	SQL_DESC_UNNAMED
SQL_DESC_LITERAL_PREFIX	SQL_DESC_UNSIGNED
SQL_DESC_LITERAL_SUFFIX	SQL_DESC_UPDATABLE

Deferred Fields:

Deferred fields are created when the descriptor header or a descriptor record is created. The addresses of the defined variables are stored but not used until a later point in the application. The application must not deallocate or discard these variables between the time it associates them with the fields and the time CLI reads or writes them.

The following table lists the deferred fields and the meaning or a null pointer where applicable:

Table 12. Deferred Fields

Field	Meaning of Null value
SQL_DESC_DATA_PTR	The record is unbound.
SQL_DESC_INDICATOR_PTR	(none)
SQL_DESC_OCTET_LENGTH_PTR (ARD and APD only)	• ARD: The length information for that column is not returned. • APD: If the parameter is a character string, the driver assumes that string is null-terminated. For output parameters, a null value in this field prevents the driver from returning length information. (If the SQL_DESC_TYPE field does not indicate a character-string parameter, the SQL_DESC_OCTET_LENGTH_PTR field is ignored.)
SQL_DESC_ARRAY_STATUS_PTR (multirow fetch only)	A multirow fetch failed to return this component of the per-row diagnostic information.
SQL_DESC_ROWS_PROCESSED_PTR (multirow fetch only)	(none)

Bound Descriptor Records:

The SQL_DESC_DATA_PTR field in each descriptor record points to a variable that contains the parameter value (for APDs) or the column value (for ARDs). This is a deferred field that defaults to null. Once the column or parameter is bound it points to the parameter or column value. At this point the descriptor record is said to be bound.

Application Parameter Descriptors (APD)
Each bound record constitutes a bound parameter. The application must bind a parameter for each input and output parameter marker in the SQL statement before the statement is executed.

Application Row Descriptors (ARD)
Each bound record relates to a bound column.

Related concepts:

- "Consistency Checks for Descriptors in CLI Applications" on page 181
- "Descriptor Allocation and Freeing" on page 182
- "Descriptor Manipulation with Descriptor Handles in CLI Applications" on page 186
- "Descriptor Manipulation Without Using Descriptor Handles in CLI Applications" on page 188

Related reference:

- "SQLGetDescField Function (CLI) - Get Single Field Settings of Descriptor Record" in the *CLI Guide and Reference, Volume 2*
- "SQLSetDescField Function (CLI) - Set a Single Field of a Descriptor Record" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "Descriptor Header and Record Field Initialization Values (CLI)" in the *CLI Guide and Reference, Volume 2*
- "Descriptor FieldIdentifier Argument Values (CLI)" in the *CLI Guide and Reference, Volume 2*

Consistency Checks for Descriptors in CLI Applications

A consistency check is performed automatically whenever an application sets the SQL_DESC_DATA_PTR field of the APD or ARD. The check ensures that various fields are consistent with each other, and that appropriate data types have been specified. Calling `SQLSetDescRec()` always prompts a consistency check. If any of the fields is inconsistent with other fields, `SQLSetDescRec()` will return SQLSTATE HY021, "Inconsistent descriptor information."

To force a consistency check of IPD fields, the application can set the SQL_DESC_DATA_PTR field of the IPD. This setting is only used to force the consistency check. The value is not stored and cannot be retrieved by a call to `SQLGetDescField()` or `SQLGetDescRec()`.

A consistency check cannot be performed on an IRD.

Application Descriptors:

Whenever an application sets the SQL_DESC_DATA_PTR field of an APD, ARD, or IPD, DB2 CLI checks that the value of the SQL_DESC_TYPE field and the values applicable to that SQL_DESC_TYPE field are valid and consistent. This check is always performed when `SQLBindParameter()` or

`SQLBindCol()` is called, or when `SQLSetDescRec()` is called for an APD, ARD, or IPD. This consistency check includes the following checks on application descriptor fields:

- The SQL_DESC_TYPE field must be one of the valid C or SQL types. The SQL_DESC_CONCISE_TYPE field must be one of the valid C or SQL types.
- If the SQL_DESC_TYPE field indicates a numeric type, the SQL_DESC_PRECISION and SQL_DESC_SCALE fields are verified to be valid.
- If the SQL_DESC_CONCISE_TYPE field is a time data type the SQL_DESC_PRECISION field is verified to be a valid seconds precision.

The SQL_DESC_DATA_PTR field of an IPD is not normally set; however, an application can do so to force a consistency check of IPD fields. A consistency check cannot be performed on an IRD. The value that the SQL_DESC_DATA_PTR field of the IPD is set to is not actually stored, and cannot be retrieved by a call to `SQLGetDescField()` or `SQLGetDescRec()`; the setting is made only to force the consistency check.

Related concepts:

- "Descriptors in CLI Applications" on page 177

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLSetDescRec Function (CLI) - Set Multiple Descriptor Fields for a Column or Parameter Data" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "Descriptor Header and Record Field Initialization Values (CLI)" in the *CLI Guide and Reference, Volume 2*
- "Descriptor FieldIdentifier Argument Values (CLI)" in the *CLI Guide and Reference, Volume 2*

Descriptor Allocation and Freeing

Descriptors are allocated in one of two ways:

Implicitly Allocated Descriptors
When a statement handle is allocated, a set of four descriptors are implicitly allocated. When the statement handle is freed, all implicitly allocated descriptors on that handle are freed as well.

To obtain handles to these implicitly allocated descriptors an application can call `SQLGetStmtAttr()`, passing the statement handle and an *Attribute* value of:
- SQL_ATTR_APP_PARAM_DESC (APD)
- SQL_ATTR_APP_ROW_DESC (ARD)
- SQL_ATTR_IMP_PARAM_DESC (IPD)
- SQL_ATTR_IMP_ROW_DESC (IRD)

For example, the following gives access to the statement's implicitly allocated implementation parameter descriptor:

```
/* dbuse. c */
/* ... */
sqlrc = SQLGetStmtAttr ( hstmt,
                         SQL_ATTR_IMP_PARAM_DESC,
                         &hIPD,
                         SQL_IS_POINTER,
                         NULL);
```

Note: The descriptors whose handles are obtained in this manner will still be freed when the statement for which they were allocated is freed.

Explicitly Allocated Descriptors

An application can explicitly allocate application descriptors. It is not possible, however, to allocate implementation descriptors.

An application descriptor can be explicitly allocated any time the application is connected to the database. To explicitly allocate the application descriptor, call `SQLAllocHandle()` with a *HandleType* of SQL_HANDLE_DESC. For example, the following call explicitly allocates an application row descriptor:

```
rc = SQLAllocHandle( SQL_HANDLE_DESC, hdbc, &hARD ) ;
```

To use an explicitly allocated application descriptor instead of a statement's implicitly allocated descriptor, call `SQLSetStmtAttr()`, and pass the statement handle, the descriptor handle, and an *Attribute* value of either:
- SQL_ATTR_APP_PARAM_DESC (APD), or
- SQL_ATTR_APP_ROW_DESC (ARD)

When there are explicitly and implicitly allocated descriptors, the explicitly specified one is used. An explicitly allocated descriptor can be associated with more than one statement.

Field Initialization:

When an application row descriptor is allocated, its fields are initialized to the values listed in the descriptor header and record field initialization values documentation. The SQL_DESC_TYPE field is set to SQL_DEFAULT which provides for a standard treatment of database data for presentation to the application. The application may specify different treatment of the data by setting fields of the descriptor record.

The initial value of the SQL_DESC_ARRAY_SIZE header field is 1. To enable multirow fetch, the application can set this value in an ARD to the number of rows in a rowset.

There are no default values for the fields of an IRD. The fields are set when there is a prepared or executed statement.

The following fields of an IPD are undefined until a call to `SQLPrepare()` automatically populates them:
- SQL_DESC_CASE_SENSITIVE
- SQL_DESC_FIXED_PREC_SCALE
- SQL_DESC_TYPE_NAME
- SQL_DESC_DESC_UNSIGNED
- SQL_DESC_LOCAL_TYPE_NAME

Automatic Population of the IPD:

There are times when the application will need to discover information about the parameters of a prepared SQL statement. A good example is when a dynamically generated query is prepared; the application will not know anything about the parameters in advance. If the application enables automatic population of the IPD, by setting the SQL_ATTR_ENABLE_AUTO_IPD statement attribute to SQL_TRUE (using `SQLSetStmtAttr()`), then the fields of the IPD are automatically populated to describe the parameter. This includes the data type, precision, scale, and so on (the same information that `SQLDescribeParam()` returns). The application can use this information to determine if data conversion is required, and which application buffer is the most appropriate to bind the parameter to.

Automatic population of the IPD involves some overhead. If it is not necessary for this information to be automatically gathered by the CLI driver then the SQL_ATTR_ENABLE_AUTO_IPD statement attribute should be set to SQL_FALSE.

When automatic population of the IPD is active, each call to `SQLPrepare()` causes the fields of the IPD to be updated. The resulting descriptor information can be retrieved by calling the following functions:
- `SQLGetDescField()`
- `SQLGetDescRec()`
- `SQLDescribeParam()`

Freeing of Descriptors:

Explicitly Allocated Descriptors

When an explicitly allocated descriptor is freed, all statement handles to which the freed descriptor applied automatically revert to the original descriptors implicitly allocated for them.

Explicitly allocated descriptors can be freed in one of two ways:
- by calling `SQLFreeHandle()`with a *HandleType* of SQL_HANDLE_DESC
- by freeing the connection handle that the descriptor is associated with

Implicitly Allocated Descriptors

An implicitly allocated descriptor can be freed in one of the following ways:
- by calling `SQLDisconnect()` which drops any statements or descriptors open on the connection
- by calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_STMT to free the statement handle and all of the implicitly allocated descriptors associated with the statement

An implicitly allocated descriptor cannot be freed by calling `SQLFreeHandle()` with a *HandleType* of SQL_HANDLE_DESC.

Related concepts:
- "Descriptors in CLI Applications" on page 177

Related reference:
- "SQLFreeHandle Function (CLI) - Free Handle Resources" in the *CLI Guide and Reference, Volume 2*
- "SQLPrepare Function (CLI) - Prepare a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "Descriptor Header and Record Field Initialization Values (CLI)" in the *CLI Guide and Reference, Volume 2*
- "Descriptor FieldIdentifier Argument Values (CLI)" in the *CLI Guide and Reference, Volume 2*

Related samples:
- "dbuse.c -- How to use a database (CLI)"

Descriptor Manipulation with Descriptor Handles in CLI Applications

Descriptors can be manipulated using descriptor handles or with DB2 CLI functions that do not use descriptor handles. This topic describes accessing descriptors through descriptor handles. The handle of an explicitly allocated descriptor is returned in the *OutputHandlePtr* argument when the application calls `SQLAllocHandle()` to allocate the descriptor. The handle of an implicitly allocated descriptor is obtained by calling `SQLGetStmtAttr()` with either SQL_ATTR_IMP_PARAM_DESC or SQL_ATTR_IMP_ROW_DESC.

Retrieval of Descriptor Field Values:

The DB2 CLI function `SQLGetDescField()` can be used to obtain a single field of a descriptor record. `SQLGetDescRec()` retrieves the settings of multiple descriptor fields that affect the data type and storage of column or parameter data.

Setting of Descriptor Field Values:

Two methods are available for setting descriptor fields: one field at a time or multiple fields at once.

Setting of Individual Fields:

Some fields of a descriptor are read-only, but others can be set using the function `SQLSetDescField()`. Refer to the list of header and record fields in the descriptor FieldIdentifier values documentation.

Record and header fields are set differently using `SQLSetDescField()` as follows:

Header fields
The call to `SQLSetDescField()` passes the header field to be set and a record number of 0. The record number is ignored since there is only one header field per descriptor. In this case the record number of 0 does not indicate the bookmark field.

Record fields
The call to `SQLSetDescField()` passes the record field to be set and a record number of 1 or higher, or 0 to indicate the bookmark field.

The application must follow the sequence of setting descriptor fields described in the SQLSetDescField() documentation when setting individual fields of a descriptor. Setting some fields will cause DB2 CLI to automatically set other fields. A consistency check will take place after the application follows the defined steps. This will ensure that the values in the descriptor fields are consistent.

If a function call that would set a descriptor fails, the content of the descriptor fields are undefined after the failed function call.

Setting of Multiple Fields:

A predefined set of descriptor fields can be set with one call rather than setting individual fields one at a time. `SQLSetDescRec()` sets the following fields for a single column or parameter:
- SQL_DESC_TYPE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_DATA_PTR
- SQL_DESC_OCTET_LENGTH_PTR
- SQL_DESC_INDICATOR_PTR

(SQL_DESC_DATETIME_INTERVAL_CODE is also defined by ODBC but is not supported by DB2 CLI.)

For example, all of the descriptor fields listed above are set with the following call:

```
/* dbuse.c */
/* ... */
rc = SQLSetDescRec(hARD, 1, type, 0,
                  length, 0, 0, &id_no, &datalen, NULL);
```

Copying of Descriptors:

One benefit of descriptors is the fact that a single descriptor can be used for multiple purposes. For instance, an ARD on one statement handle can be used as an APD on another statement handle.

There will be other instances, however, where the application will want to make a copy of the original descriptor, then modify certain fields. In this case `SQLCopyDesc()` is used to overwrite the fields of an existing descriptor with the values from another descriptor. Only fields that are defined for both the source and target descriptors are copied (with the exception of the SQL_DESC_ALLOC_TYPE field which cannot be changed).

Fields can be copied from any type of descriptor, but can only be copied to an application descriptor (APD or ARD) or an IPD. Fields cannot be copied to an IRD. The descriptor's allocation type will not be changed by the copy procedure (again, the SQL_DESC_ALLOC_TYPE field cannot be changed).

Related concepts:
- "Handles in CLI" on page 19

- “Descriptors in CLI Applications” on page 177
- “Consistency Checks for Descriptors in CLI Applications” on page 181

Related reference:
- “SQLGetDescField Function (CLI) - Get Single Field Settings of Descriptor Record” in the *CLI Guide and Reference, Volume 2*
- “SQLGetDescRec Function (CLI) - Get Multiple Field Settings of Descriptor Record” in the *CLI Guide and Reference, Volume 2*
- “SQLGetStmtAttr Function (CLI) - Get Current Setting of a Statement Attribute” in the *CLI Guide and Reference, Volume 2*
- “SQLSetDescField Function (CLI) - Set a Single Field of a Descriptor Record” in the *CLI Guide and Reference, Volume 2*
- “SQLSetDescRec Function (CLI) - Set Multiple Descriptor Fields for a Column or Parameter Data” in the *CLI Guide and Reference, Volume 2*
- “Statement Attributes (CLI) List” in the *CLI Guide and Reference, Volume 2*
- “Descriptor Header and Record Field Initialization Values (CLI)” in the *CLI Guide and Reference, Volume 2*
- “Descriptor FieldIdentifier Argument Values (CLI)” in the *CLI Guide and Reference, Volume 2*

Related samples:
- “dbuse.c -- How to use a database (CLI)”

Descriptor Manipulation Without Using Descriptor Handles in CLI Applications

Many CLI functions make use of descriptors, but the application itself does not need to manipulate them directly. Instead, the application can use a different function which will set or retrieve one or more fields of a descriptor as well as perform other functions. This category of CLI functions is called *concise* functions. `SQLBindCol()` is an example of a concise function that manipulates descriptor fields.

In addition to manipulating multiple fields, concise functions are called without explicitly specifying the descriptor handle. The application does not even need to retrieve the descriptor handle to use a concise function.

The following types of concise functions exist:
- The functions `SQLBindCol()` and `SQLBindParameter()` bind a column or parameter by setting the descriptor fields that correspond to their arguments. These functions also perform other tasks unrelated to descriptors.

If required, an application can also use the descriptor calls directly to modify individual details of a binding. In this case the descriptor handle must be retrieved, and the functions `SQLSetDescField()` or `SQLSetDescRec()` are called to modify the binding.

- The following functions always retrieve values in descriptor fields:
  - `SQLColAttribute()`
  - `SQLDescribeCol()`
  - `SQLDescribeParam()`
  - `SQLNumParams()`
  - `SQLNumResultCols()`
- The functions `SQLSetDescRec()` and `SQLGetDescRec()` set or get the multiple descriptor fields that affect the data type and storage of column or parameter data. A single call to `SQLSetDescRec()` can be used to change the values used in the binding of a column or parameter.
- The functions `SQLSetStmtAttr()` and `SQLGetStmtAttr()` modify or return descriptor fields in some cases, depending on which statement attribute is specified. Refer to the "Values Stored in a Descriptor" section of the descriptors documentation for more information.

Related concepts:

- "Handles in CLI" on page 19
- "Descriptors in CLI Applications" on page 177

Related reference:

- "SQLBindCol Function (CLI) - Bind a Column to an Application Variable or LOB Locator" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDescRec Function (CLI) - Get Multiple Field Settings of Descriptor Record" in the *CLI Guide and Reference, Volume 2*
- "SQLGetStmtAttr Function (CLI) - Get Current Setting of a Statement Attribute" in the *CLI Guide and Reference, Volume 2*
- "SQLSetDescField Function (CLI) - Set a Single Field of a Descriptor Record" in the *CLI Guide and Reference, Volume 2*
- "SQLSetDescRec Function (CLI) - Set Multiple Descriptor Fields for a Column or Parameter Data" in the *CLI Guide and Reference, Volume 2*
- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "SQLBindParameter Function (CLI) - Bind A Parameter Marker to a Buffer or LOB Locator" in the *CLI Guide and Reference, Volume 2*

Chapter 15. Environment, Connection, and Statement Attributes

Environment, Connection, and Statement Attributes in CLI Applications

Environments, connections, and statements each have a defined set of attributes (or options). All attributes can be queried by the application, but only some attributes can be changed from their default values. By changing attribute values, the application can change the behavior of DB2 CLI.

An environment handle has attributes which affect the behavior of DB2 CLI functions under that environment. The application can specify the value of an attribute by calling `SQLSetEnvAttr()` and can obtain the current attribute value by calling `SQLGetEnvAttr()`. `SQLSetEnvAttr()` can only be called before any connection handles have been allocated for the environment handle. For details on each environment attribute, refer to the list of CLI environment attributes.

A connection handle has attributes which affect the behavior of DB2 CLI functions under that connection. Of the attributes that can be changed:
- Some can be set any time once the connection handle is allocated.
- Some can be set only before the actual connection has been established.
- Some can be set any time after the connection has been established.
- Some can be set after the connection has been established, but only while there are no outstanding transactions or open cursors.

The application can change the value of connection attributes by calling `SQLSetConnectAttr()` and can obtain the current value of an attribute by calling `SQLGetConnectAttr()`. An example of a connection attribute which can be set any time after a handle is allocated is the auto-commit option SQL_ATTR_MAXCONN. For details on each connection attribute, refer to the list of CLI connection attributes.

A statement handle has attributes which affect the behavior of CLI functions executed using that statement handle. Of the statement attributes that can be changed:
- Some attributes can be set, but currently are limited to only one specific value.
- Some attributes can be set any time after the statement handle has been allocated.
- Some attributes can only be set if there is no open cursor on that statement handle.

The application can specify the value of any statement attribute that can be set by calling `SQLSetStmtAttr()` and can obtain the current value of an attribute by calling `SQLGetStmtAttr()`. For details on each statement attribute, refer to the list of CLI statement attributes.

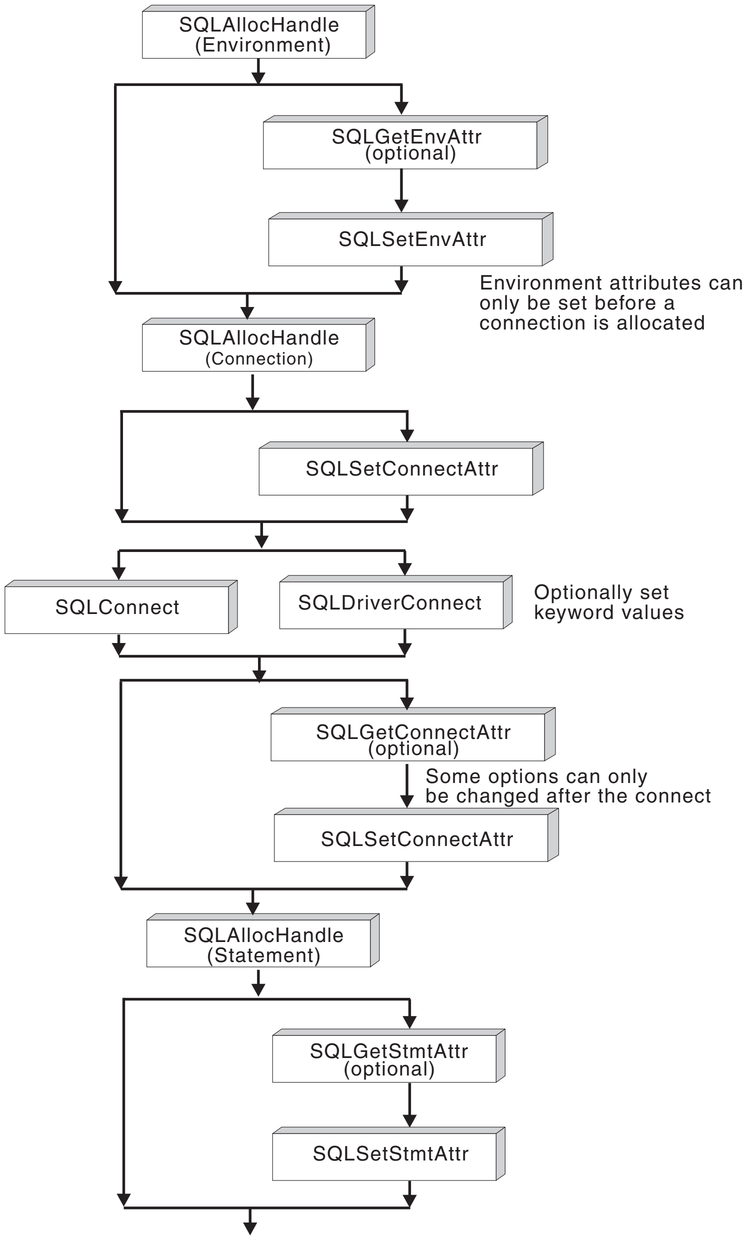
The `SQLSetConnectAttr()` function cannot be used to set statement attributes. This was supported in versions of DB2 CLI prior to version 5.

Many applications just use the default attribute settings; however, there may be situations where some of these defaults are not suitable for a particular user of the application. Some default values can be changed by setting the CLI/ODBC configuration keywords. DB2 CLI provides end users with two methods of setting some configuration keywords. The first method is to specify the keyword and its new default attribute value(s) in the connection string input to the `SQLDriverConnect()` and `SQLBrowseConnect()` functions. The second method involves the specification of the new default attribute value(s) in a DB2 CLI initialization file using CLI/ODBC configuration keywords.

The DB2 CLI initialization file can be used to change default values for all DB2 CLI applications on that workstation. This may be the end user's only means of changing the defaults if the application does not provide a means for the user to provide default attribute values in the `SQLDriverConnect()` connection string. Default attribute values that are specified on `SQLDriverConnect()` override the values in the DB2 CLI initialization file for that particular connection.

The mechanisms for changing defaults are intended for end user tuning; application developers must use the appropriate set-attribute function. If an application does call a set-attribute or option function with a value different from the initialization file or the connection string specification, then the initial default value is overridden and the new value takes effect.

The diagram below shows the addition of the attribute functions to the basic connect scenario.



SQLAllocHandle (Environment)
SQLGetEnvAttr (optional)
SQLSetEnvAttr
Environment attributes can only be set before a connection is allocated
SQLAllocHandle (Connection)
SQLSetConnectAttr
SQLConnect
SQLDriverConnect
Optionally set keyword values
SQLGetConnectAttr (optional)
Some options can only be changed after the connect
SQLSetConnectAttr
SQLAllocHandle (Statement)
SQLGetStmtAttr (optional)
SQLSetStmtAttr

Figure 12. Setting and Retrieving Attributes (Options)

Related concepts:

- "Programming Hints and Tips for CLI Applications" on page 65
- "Handles in CLI" on page 19
- "db2cli.ini Initialization File" on page 293

Related reference:

- "SQLGetConnectAttr Function (CLI) - Get Current Attribute Setting" in the *CLI Guide and Reference, Volume 2*
- "SQLGetEnvAttr Function (CLI) - Retrieve Current Environment Attribute Value" in the *CLI Guide and Reference, Volume 2*
- "SQLSetConnectAttr Function (CLI) - Set Connection Attributes" in the *CLI Guide and Reference, Volume 2*
- "SQLSetEnvAttr Function (CLI) - Set Environment Attribute" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "Environment Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

Related samples:

- "dbuse.c -- How to use a database (CLI)"
- "spcall.c -- Call individual stored procedures (CLI)"
- "tbread.c -- How to read data from tables (CLI)"
- "tut_use.c -- How to execute SQL statements, bind parameters to an SQL statement (CLI)"

Chapter 16. Querying System Catalog Information

Catalog Functions for Querying System Catalog Information in CLI Applications

One of the first tasks an application often performs is to display a list of tables from which one or more are selected by the user. Although the application can issue its own queries against the database system catalog to get catalog information for such a DB2 command, it is best that the application calls the DB2 CLI catalog functions instead. These catalog functions, also called schema functions, provide a generic interface to issue queries and return consistent result sets across the DB2 family of servers. This allows the application to avoid server-specific and release-specific catalog queries.

The catalog functions operate by returning to the application a result set through a statement handle. Calling these functions is conceptually equivalent to using `SQLExecDirect()` to execute a select against the system catalog tables. After calling these functions, the application can fetch individual rows of the result set as it would process column data from an ordinary `SQLFetch()`. The DB2 CLI catalog functions are:

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLForeignKeys()`
- `SQLGetTypeInfo()`
- `SQLPrimaryKeys()`
- `SQLProcedureColumns()`
- `SQLProcedures()`
- `SQLSpecialColumns()`
- `SQLStatistics()`
- `SQLTablePrivileges()`
- `SQLTables()`

The result sets returned by these functions are defined in the descriptions for each catalog function. The columns are defined in a specified order. In future releases, other columns may be added to the end of each defined result set, therefore applications should be written in a way that would not be affected by such changes.

Some of the catalog functions result in execution of fairly complex queries, and for this reason should only be called when needed. It is recommended that the application save the information returned rather than making repeated calls to get the same information.

Related concepts:

- "Input Arguments on Catalog Functions in CLI Applications" on page 196

Related reference:

- "CLI and ODBC Function Summary" in the *CLI Guide and Reference, Volume 2*

Input Arguments on Catalog Functions in CLI Applications

All of the catalog functions have *CatalogName* and *SchemaName* (and their associated lengths) on their input argument list. Other input arguments may also include *TableName*, *ProcedureName*, or *ColumnName* (and their associated lengths). These input arguments are used to either identify or constrain the amount of information to be returned.

In the Function Arguments section for the catalog functions each of the above input arguments are described either as a pattern value or as an ordinary non-pattern matching argument. For example, `SQLColumnPrivileges()` treats *SchemaName* and *TableName* as ordinary arguments and *ColumnName* as a pattern value.

An input treated as an ordinary argument is treated as a literal and the case of letters is significant. These arguments limit the scope of the query by identifying the object of interest. An error results if the application passes a null pointer for the argument.

Inputs treated as pattern values are used to constrain the size of the result set by including only matching rows as though the underlying query's WHERE clause contained a LIKE predicate. If the application passes a null pointer for a pattern value input, the argument is not used to restrict the result set (that is, there is no corresponding LIKE in the WHERE clause). If a catalog function has more than one pattern value input argument, they are treated as though the LIKE predicates of the WHERE clauses in the underlying query were joined by AND; a row appears in this result set only if it meets all the conditions of the LIKE predicates.

Each pattern value argument can contain:

- The underscore (_) character which stands for any single character.
- The percent (%) character which stands for any sequence of zero or more characters. Note that providing a pattern value containing a single % is equivalent to passing a null pointer for that argument.
- Characters with no special meaning which stand for themselves. The case of a letter is significant.

These argument values are used on conceptual LIKE predicate(s) in the WHERE clause. To treat the metadata characters (_, %) as themselves, an escape character must immediately precede the _ or %. The escape character itself can be specified as part of the pattern by including it twice in succession. An application can determine the escape character by calling `SQLGetInfo()` with SQL_SEARCH_PATTERN_ESCAPE.

For example, the following calls would retrieve all the tables that start with 'ST':

```
/* tbinfo.c */
/* ... */
struct
{   SQLINTEGER ind ;
    SQLCHAR    val[129] ;
} tbQualifier, tbSchema, tbName, tbType;

struct
{   SQLINTEGER ind ;
    SQLCHAR val[255] ;
} tbRemarks;

SQLCHAR tbSchemaPattern[] = "%";
SQLCHAR tbNamePattern[] = "ST%";  /* all the tables starting with ST */

/* ... */
sqlrc = SQLTables( hstmt, NULL, 0,
                   tbSchemaPattern, SQL_NTS,
                   tbNamePattern, SQL_NTS,
                   NULL, 0);
/* ... */

/* bind columns to variables */
sqlrc = SQLBindCol( hstmt, 1, SQL_C_CHAR, tbQualifier.val, 129,
                    &tbQualifier.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 2, SQL_C_CHAR, tbSchema.val, 129,
                    &tbSchema.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 3, SQL_C_CHAR, tbName.val, 129,
                    &tbName.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 4, SQL_C_CHAR, tbType.val, 129,
                    &tbType.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);
sqlrc = SQLBindCol( hstmt, 5, SQL_C_CHAR, tbRemarks.val, 255,
                    &tbRemarks.ind ) ;
STMT_HANDLE_CHECK( hstmt, sqlrc);

/* ... */
sqlrc = SQLFetch( hstmt );
/* ... */
while (sqlrc != SQL_NO_DATA_FOUND)
```

```
    {   /* ... */
        sqlrc = SQLFetch( hstmt );
        /* ... */
    }
```

Related concepts:

- "Catalog Functions for Querying System Catalog Information in CLI Applications" on page 195

Related samples:

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

Chapter 17. Vendor Escape Clauses

Vendor Escape Clauses in CLI Applications

The X/Open SQL CAE specification defined an **escape clause** as: "a syntactic mechanism for vendor-specific SQL extensions to be implemented in the framework of standardized SQL". Both DB2 CLI and ODBC support vendor escape clauses as defined by X/Open.

Currently, escape clauses are used extensively by ODBC to define SQL extensions. DB2 CLI translates the ODBC extensions into the correct DB2 syntax. The `SQLNativeSql()` function can be used to display the resulting syntax.

If an application is only going to access DB2 data sources, then there is no reason to use the escape clauses. If an application is going to access other data sources that offer the same support through a different syntax, then the escape clauses increase the portability of the application.

DB2 CLI used both the standard and shorthand syntax for escape clauses. The standard syntax has been deprecated (although DB2 CLI still supports it). An escape clause using the standard syntax took the form:

```
--(*vendor(vendor-identifier),
     product(product-identifier) extended SQL text*)--
```

Applications should now only use the shorthand syntax, as described below, to remain current with the latest ODBC standards.

Shorthand Escape Clause Syntax:

The format of an escape clause definition is:

```
{ extended SQL text }
```

to define the following SQL extensions:

- Extended date, time, timestamp data
- Outer join
- LIKE predicate
- Stored procedure call
- Extended scalar functions
  - Numeric functions
  - String functions

– System functions

ODBC Date, Time, Timestamp Data:

The ODBC escape clauses for date, time, and timestamp data are:

```
{d 'value'}
{t 'value'}
{ts 'value'}
```

d indicates *value* is a date in the *yyyy-mm-dd* format,
t indicates *value* is a time in the *hh:mm:ss* format
ts indicates *value* is a timestamp in the *yyyy-mm-dd hh:mm:ss[.f...]* format.

For example, the following statement can be used to issue a query against the **EMPLOYEE** table:

```
SELECT * FROM EMPLOYEE WHERE HIREDATE={d '1994-03-29'}
```

DB2 CLI will translate the above statement to a DB2 format. `SQLNativeSql()` can be used to return the translated statement.

The ODBC escape clauses for date, time, and timestamp literals can be used in input parameters with a C data type of SQL_C_CHAR.

ODBC Outer Join:

The ODBC escape clause for outer join is:

```
{oj outer-join}
```

where *outer join* is

```
  table-name  {LEFT | RIGHT | FULL} OUTER JOIN
              {table-name | outer-join}
              ON search-condition
```

For example, DB2 CLI will translate the following statement:

```
SELECT * FROM {oj T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C3}
    WHERE T1.C2>20
```

to IBM's format, which corresponds to the SQL92 outer join syntax:

```
SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1=T2.C3 WHERE T1.C2>20
```

Note: Not all DB2 servers support outer join. To determine if the current server supports outer joins, call `SQLGetInfo()` with the SQL_SQL92_RELATIONAL_JOIN_OPERATORS and SQL_OJ_CAPABILITIES options.

LIKE Predicate:

In a SQL LIKE predicate, the metacharacter % matches zero or more of any character, and the metacharacter _ matches any one character. The SQL ESCAPE clause allows the definition of patterns intended to match values that contain the actual percent and underscore characters by preceding them with an escape character. The escape clause ODBC uses to define the LIKE predicate escape character is:

```
{escape 'escape-character'}
```

where *escape-character* is any character supported by the DB2 rules governing the use of the SQL ESCAPE clause.

As an example of how to use an "escape" ODBC escape clause, suppose you had a table Customers with the columns Name and Growth. The Growth column contains data having the metacharacter '%'. The following statement would select all of the values from Name that have values in Growth only between 10% and 19%, excluding 100% and above:

```
SELECT Name FROM Customers WHERE Growth LIKE '1_\%'{escape '\'}
```

Applications that are not concerned about portability across different vendor DBMS products should pass an SQL ESCAPE clause directly to the data source. To determine when LIKE predicate escape characters are supported by a particular DB2® data source, an application should call `SQLGetInfo()` with the SQL_LIKE_ESCAPE_CLAUSE information type.

Stored Procedure Call:

The ODBC escape clause for calling a stored procedure is:

```
 {[?=]call procedure-name[([parameter][,[parameter]]...)]}
```

where:
- [?=] indicates the optional parameter marker for the return value
- *procedure-name* specifies the name of a procedure stored at the data source
- *parameter* specifies a procedure parameter.

A procedure may have zero or more parameters.

ODBC specifies the optional parameter **?=** to represent the procedure's return value, which if present, will be stored in the location specified by the first parameter marker as defined via `SQLBindParameter()`. DB2 CLI will return the SQLCODE as the procedure's return value if **?=** is present in the escape clause. If **?=** is not present, then the application can retrieve the SQLCODE from the SQLCA by using the `SQLGetSQLCA()` function. DB2 CLI supports literals as procedure arguments, however vendor escape clauses must be used. For example, the following statement would not succeed: `CALL storedproc` `('aaaa', 1)`, but this statement would: `{CALL storedproc ('aaaa', 1)}`. If a parameter is an output parameter, it must be a parameter marker.

For example, DB2 CLI will translate the following statement:

```
{CALL NETB94(?,?,?)}
```

To an internal CALL statement format:

```
CALL NEBT94(?, ?, ?)
```

ODBC Scalar Functions:

Scalar functions such as string length, substring, or trim can be used on columns of a result set and on columns that restrict rows of a result set. The ODBC escape clause for scalar functions is:

```
{fn scalar-function}
```

Where, *scalar-function* can be any function listed in the list of extended scalar functions.

For example, DB2 CLI will translate the following statement:

```
SELECT {fn CONCAT(FIRSTNAME,LASTNAME)} FROM EMPLOYEE
```

to:

```
SELECT FIRSTNAME CONCAT LASTNAME FROM EMPLOYEE
```

`SQLNativeSql()` can be called to obtain the translated SQL statement.

To determine which scalar functions are supported by the current server referenced by a specific connection handle, call `SQLGetInfo()` with the options: SQL_NUMERIC_FUNCTIONS, SQL_STRING_FUNCTIONS, SQL_SYSTEM_FUNCTIONS, and SQL_TIMEDATE_FUNCTIONS.

Related reference:

- "SQLGetInfo Function (CLI) - Get General Information" in the *CLI Guide and Reference, Volume 2*
- "SQLNativeSql Function (CLI) - Get Native SQL Text" in the *CLI Guide and Reference, Volume 2*
- "Extended Scalar Functions for CLI Applications" on page 203
- "LIKE predicate" in the *SQL Reference, Volume 1*
- "CALL statement" in the *SQL Reference, Volume 2*
- "SELECT statement" in the *SQL Reference, Volume 2*

Related samples:

- “dbnative.c -- How to translate a statement that contains an ODBC escape clause (CLI)”

Extended Scalar Functions for CLI Applications

The following functions are defined by ODBC using vendor escape clauses. Each function may be called using the escape clause syntax, or calling the equivalent DB2 function.

These functions are presented in the following categories:
- “String Functions”
- “Numeric Functions” on page 206
- “Date and Time Functions” on page 209
- “System Functions” on page 213
- “Conversion Function” on page 213

The tables in the following sections indicates for which servers (and the earliest versions) that the function can be accessed, when called from an application using DB2 CLI

All errors detected by the following functions, when connected to a DB2 Version 5 or later server, will return SQLSTATE 38552. The text portion of the message is of the form SYSFUN:*nn* where *nn* is one of the following reason codes:

01	Numeric value out of range
02	Division by zero
03	Arithmetic overflow or underflow
04	Invalid date format
05	Invalid time format
06	Invalid timestamp format
07	Invalid character representation of a timestamp duration
08	Invalid interval type (must be one of 1, 2, 4, 8, 16, 32, 64, 128, 256)
09	String too long
10	Length or position in string function out of range
11	Invalid character representation of a floating point number

String Functions:

The string functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.
- Character string literals used as arguments to scalar functions must be bounded by single quotes.

- Arguments denoted as *string_exp* can be the name of a column, a string literal, or the result of another scalar function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, or SQL_CLOB.
- Arguments denoted as *start, length, code* or *count* can be a numeric literal or the result of another scalar function, where the underlying data type is integer based (SQL_SMALLINT, SQL_INTEGER).
- The first character in the string is considered to be at position 1.

Table 13. String Scalar Functions

ASCII(*string_exp* **)** Returns the ASCII code value of the leftmost character of *string_exp* as an integer.				
DB2 for	workstation			
CHAR(*code* **)** Returns the character that has the ASCII code value specified by *code*. The value of *code* should be between 0 and 255; otherwise, the return value is null.				
DB2 for	workstation			
CONCAT(*string_exp1***,** *string_exp2* **)** Returns a character string that is the result of concatenating *string_exp2* to *string_exp1*.				
DB2 for	workstation	MVS	VM/VSE	AS/400
DIFFERENCE(*string_exp1, string_exp2* **)** Returns an integer value indicating the difference between the values returned by the SOUNDEX function for *string_exp1* and *string_exp2*.				
DB2 for	workstation			
INSERT(*string_exp1, start, length, string_exp2* **)** Returns a character string where *length* number of characters beginning at *start* has been replaced by *string_exp2* which contains *length* characters.				
DB2 for	workstation	MVS	VM/VSE	AS/400
LCASE(*string_exp* **)** Converts all upper case characters in *string_exp* to lower case.				
DB2 for	workstation		VM/VSE	
LEFT(*string_exp,count* **)** Returns the leftmost *count* of characters of *string_exp*.				
DB2 for	workstation	MVS	VM/VSE	AS/400

Table 13. String Scalar Functions (continued)

LENGTH(*string_exp* **)** Returns the number of characters in *string_exp*, excluding trailing blanks and the string termination character. **Note:** Trailing blanks are included for DB2 for MVS/ESA.				
DB2 for	workstation	MVS	VM/VSE	AS/400
LOCATE(*string_exp1*, *string_exp2* **[** **,***start* **])** Returns the starting position of the first occurrence of *string_exp1* within *string_exp2*. The search for the first occurrence of *string_exp1* begins with first character position in *string_exp2* unless the optional argument, *start*, is specified. If *start* is specified, the search begins with the character position indicated by the value of *start*. The first character position in *string_exp2* is indicated by the value 1. If *string_exp1* is not found within *string_exp2*, the value 0 is returned.				
DB2 for	workstation			
LTRIM(*string_exp* **)** Returns the characters of *string_exp* with the leading blanks removed.				
DB2 for	workstation	VM/VSE		AS/400
REPEAT(*string_exp*, *count* **)** Returns a character string composed of *string_exp* repeated *count* times.				
DB2 for	workstation	MVS	VM/VSE	AS/400
REPLACE(*string_exp1*, *string_exp2*, *string_exp3* **)** Replaces all occurrences of *string_exp2* in *string_exp1* with *string_exp3*.				
DB2 for	workstation			
RIGHT(*string_exp*, *count* **)** Returns the rightmost count of characters of *string_exp*.				
DB2 for	workstation	MVS	VM/VSE	AS/400
RTRIM(*string_exp* **)** Returns the characters of *string_exp* with trailing blanks removed.				
DB2 for	workstation		VM/VSE	AS/400
SOUNDEX(*string_exp1* **)** Returns a four character code representing the sound of *string_exp1*. Note that different data sources use different algorithms to represent the sound of *string_exp1*.				
DB2 for	workstation			
SPACE(*count* **)** Returns a character string consisting of *count* spaces.				

Table 13. String Scalar Functions (continued)

DB2 for	workstation			
SUBSTRING(*string_exp*, *start*, *length* **)** Returns a character string that is derived from *string_exp* beginning at the character position specified by *start* for *length* characters.				
DB2 for	workstation	MVS	VM/VSE	AS/400
UCASE(*string_exp* **)** Converts all lower case characters in *string_exp* to upper case.				
DB2 for	workstation		VM/VSE	AS/400

Numeric Functions:

The numeric functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Arguments denoted as *numeric_exp* can be the name of a column, the result of another scalar function, or a numeric literal, where the underlying data type can be either floating point based (SQL_NUMERIC, SQL_DECIMAL, SQL_FLOAT, SQL_REAL, SQL_DOUBLE) or integer based (SQL_SMALLINT, SQL_INTEGER).
- Arguments denoted as *double_exp* can be the name of a column, the result of another scalar functions, or a numeric literal where the underlying data type is floating point based.
- Arguments denoted as *integer_exp* can be the name of a column, the result of another scalar functions, or a numeric literal, where the underlying data type is integer based.

Table 14. Numeric Scalar Functions

ABS(*numeric_exp* **)** Returns the absolute value of *numeric_exp*.				
DB2 for	workstation			AS/400
ACOS(*double_exp* **)** Returns the arccosine of *double_exp* as an angle, expressed in radians.				
DB2 for	workstation			AS/400
ASIN(*double_exp* **)** Returns the arcsine of *double_exp* as an angle, expressed in radians.				
DB2 for	workstation			AS/400
ATAN(*double_exp* **)** Returns the arctangent of *double_exp* as an angle, expressed in radians.				

Table 14. Numeric Scalar Functions (continued)

DB2 for	workstation			AS/400
ATAN2(*double_exp1***,** *double_exp2* **)** Returns the arctangent of *x* and *y* coordinates specified by *double_exp1* and *double_exp2*, respectively, as an angle expressed in radians.				
DB2 for	workstation			
CEILING(*numeric_exp* **)** Returns the smallest integer greater than or equal to *numeric_exp*.				
DB2 for	workstation			
COS(*double_exp* **)** Returns the cosine of *double_exp*, where *double_exp* is an angle expressed in radians.				
DB2 for	workstation			AS/400
COT(*double_exp* **)** Returns the cotangent of *double_exp*, where *double_exp* is an angle expressed in radians.				
DB2 for	workstation			AS/400
DEGREES(*numeric_exp* **)** Returns the number of degrees converted from *numeric_exp* radians.				
DB2 for	workstation			AS/400 3.6
EXP(*double_exp* **)** Returns the exponential value of *double_exp*.				
DB2 for	workstation			AS/400
FLOOR(*numeric_exp* **)** Returns the largest integer less than or equal to *numeric_exp*.				
DB2 for	workstation			AS/400 3.6
LOG(*double_exp* **)** Returns the natural logarithm of *double_exp*.				
DB2 for	workstation			AS/400
LOG10(*double_exp* **)** Returns the base 10 logarithm of *double_exp*.				
DB2 for	workstation			AS/400
MOD(*integer_exp1***,** *integer_exp2* **)** Returns the remainder (modulus) of *integer_exp1* divided by *integer_exp2*.				
DB2 for	workstation			AS/400

Table 14. Numeric Scalar Functions (continued)

PI() Returns the constant value of pi as a floating point value.				
DB2 for	workstation			AS/400
POWER(*numeric_exp*, *integer_exp* **)** Returns the value of *numeric_exp* to the power of *integer_exp*.				
DB2 for	workstation			AS/400 3.6
RADIANS(*numeric_exp* **)** Returns the number of radians converted from *numeric_exp* degrees.				
DB2 for	workstation			
RAND([*integer_exp* **])** Returns a random floating point value using *integer_exp* as the optional seed value.				
DB2 for	workstation			
ROUND(*numeric_exp*, *integer_exp.* **)** Returns *numeric_exp* rounded to *integer_exp* places right of the decimal point. If *integer_exp* is negative, *numeric_exp* is rounded to | *integer_exp* | places to the left of the decimal point.				
DB2 for	workstation			
SIGN(*numeric_exp* **)** Returns an indicator or the sign of *numeric_exp*. If *numeric_exp* is less than zero, -1 is returned. If *numeric_exp* equals zero, 0 is returned. If *numeric_exp* is greater than zero, 1 is returned.				
DB2 for	workstation			
SIN(*double_exp* **)** Returns the sine of *double_exp*, where *double_exp* is an angle expressed in radians.				
DB2 for	workstation			AS/400
SQRT(*double_exp* **)** Returns the square root of *double_exp*.				
DB2 for	workstation			AS/400
TAN(*double_exp* **)** Returns the tangent of *double_exp*, where *double_exp* is an angle expressed in radians.				
DB2 for	workstation			AS/400

Table 14. Numeric Scalar Functions (continued)

TRUNCATE(*numeric_exp*, *integer_exp* **)** Returns *numeric_exp* truncated to *integer_exp* places right of the decimal point. If *integer_exp* is negative, *numeric_exp* is truncated to ǀ *integer_exp* ǀ places to the left of the decimal point.				
DB2 for	workstation			

Date and Time Functions:

The date and time functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Arguments denoted as *timestamp_exp* can be the name of a column, the result of another scalar function, or a time, date, or timestamp literal.
- Arguments denoted as *date_exp* can be the name of a column, the result of another scalar function, or a date or timestamp literal, where the underlying data type can be character based, or date or timestamp based.
- Arguments denoted as *time_exp* can be the name of a column, the result of another scalar function, or a time or timestamp literal, where the underlying data types can be character based, or time or timestamp based.

Table 15. Date and Time Scalar Functions

CURDATE() Returns the current date as a date value.				
DB2 for	workstation	MVS	VM/VSE	AS/400
CURTIME() Returns the current local time as a time value.				
DB2 for	workstation	MVS	VM/VSE	AS/400
DAYNAME(*date_exp* **)** Returns a character string containing the name of the day (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday) for the day portion of *date_exp*.				
DB2 for	workstation			
DAYOFMONTH (*date_exp* **)** Returns the day of the month in *date_exp* as an integer value in the range of 1-31.				
DB2 for	workstation	MVS	VM/VSE	AS/400
DAYOFWEEK(*date_exp* **)** Returns the day of the week in *date_exp* as an integer value in the range 1-7, where 1 represents Sunday.				
DB2 for	workstation			AS/400 3.6

Table 15. Date and Time Scalar Functions (continued)

DAYOFWEEK_ISO(*date_exp* **)** Returns the day of the week in *date_exp* as an integer value in the range 1-7, where 1 represents Monday. Note the difference between this function and the DAYOFWEEK() function, where 1 represents Sunday.				
DB2 for	workstation			
DAYOFYEAR(*date_exp* **)** Returns the day of the year in *date_exp* as an integer value in the range 1-366.				
DB2 for	workstation			AS/400 3.6
HOUR(*time_exp* **)** Returns the hour in *time_exp* as an integer value in the range of 0-23.				
DB2 for	workstation	MVS	VM/VSE	AS/400
JULIAN_DAY(*date_exp* **)** Returns the number of days between *date_exp* and January 1, 4712 B.C. (the start of the Julian date calendar).				
DB2 for	workstation			
MINUTE(*time_exp* **)** Returns the minute in *time_exp* as integer value in the range of 0-59.				
DB2 for	workstation	MVS	VM/VSE	AS/400
MONTH(*date_exp* **)** Returns the month in *date_exp* as an integer value in the range of 1-12.				
DB2 for	workstation	MVS	VM/VSE	AS/400
MONTHNAME(*date_exp* **)** Returns a character string containing the name of month (January, February, March, April, May, June, July, August, September, October, November, December) for the month portion of *date_exp*.				
DB2 for	workstation			
NOW() Returns the current date and time as a timestamp value.				
DB2 for	workstation	MVS	VM/VSE	AS/400
QUARTER(*date_exp* **)** Returns the quarter in *date_exp* as an integer value in the range of 1-4.				
DB2 for	workstation			AS/400 3.6
SECOND(*time_exp* **)** Returns the second in *time_exp* as an integer value in the range of 0-59.				

Table 15. Date and Time Scalar Functions (continued)

DB2 for	workstation	MVS	VM/VSE	AS/400
SECONDS_SINCE_MIDNIGHT(*time_exp* **)** Returns the number of seconds in *time_exp* relative to midnight as an integer value in the range of 0-86400. If *time_exp* includes a fractional seconds component, the fractional seconds component will be discarded.				
DB2 for	workstation			
TIMESTAMPADD(*interval*, *integer_exp*, *timestamp_exp* **)** Returns the timestamp calculated by adding *integer_exp* intervals of type *interval* to *timestamp_exp*. Valid values of interval are: • SQL_TSI_FRAC_SECOND • SQL_TSI_SECOND • SQL_TSI_MINUTE • SQL_TSI_HOUR • SQL_TSI_DAY • SQL_TSI_WEEK • SQL_TSI_MONTH • SQL_TSI_QUARTER • SQL_TSI_YEAR where fractional seconds are expressed in billionths of a second. If *timestamp_exp* specifies a time value and *interval* specifies days, weeks, months, quarters, or years, the date portion of *timestamp_exp* is set to the current date before calculating the resulting timestamp. If *timestamp_exp* is a date value and *interval* specifies fractional seconds, seconds, minutes, or hours, the time portion of *timestamp_exp* is set to 00:00:00.000000 before calculating the resulting timestamp. An application determines which intervals are supported by calling ***SQLGetInfo()*** with the SQL_TIMEDATE_ADD_INTERVALS option.				
DB2 for	workstation			

Table 15. Date and Time Scalar Functions (continued)

TIMESTAMPDIFF(*interval*, *timestamp_exp1*, *timestamp_exp2* **)** Returns the integer number of intervals of type *interval* by which *timestamp_exp2* is greater than *timestamp_exp1*. Valid values of interval are: • SQL_TSI_FRAC_SECOND • SQL_TSI_SECOND • SQL_TSI_MINUTE • SQL_TSI_HOUR • SQL_TSI_DAY • SQL_TSI_WEEK • SQL_TSI_MONTH • SQL_TSI_QUARTER • SQL_TSI_YEAR where fractional seconds are expressed in billionths of a second. If either timestamp expression is a time value and *interval* specifies days, weeks, months, quarters, or years, the date portion of that timestamp is set to the current date before calculating the difference between the timestamps. If either timestamp expression is a date value and *interval* specifies fractional seconds, seconds, minutes, or hours, the time portion of that timestamp is set to 0 before calculating the difference between the timestamps. An application determines which intervals are supported by calling ***SQLGetInfo()*** with the SQL_TIMEDATE_DIFF_INTERVALS option.				
DB2 for	workstation			
WEEK(*date_exp* **)** Returns the week of the year in *date_exp* as an integer value in the range of 1-54.				
DB2 for	workstation			AS/400 3.6
WEEK_ISO(*date_exp* **)** Returns the week of the year in *date_exp* as an integer value in the range of 1-53. Week 1 is defined as the first week of the year to contain a Thursday. Therefore, Week1 is equivalent to the first week that contains Jan 4, since Monday is considered to be the first day of the week. Note that WEEK_ISO() differs from the current definition of WEEK(), which returns a value up to 54. For the WEEK() function, Week 1 is the week containing the first Saturday. This is equivalent to the week containing Jan. 1, even if the week contains only one day.				
DB2 for	workstation			
YEAR(*date_exp* **)** Returns the year in *date_exp* as an integer value in the range of 1-9999.				
DB2 for	workstation	MVS	VM/VSE	AS/400

For those functions that return a character string containing the name of the day of week or the name of the month, these character strings will be National Language Support enabled.

DAYOFWEEK_ISO() and WEEK_ISO() are automatically available in a database created in DB2 Version 7 or later. If a database was created prior to Version 7, these functions may not be available. To make DAYOFWEEK_ISO() and WEEK_ISO() functions available in such a database, use the **db2updb** system command.

System Functions:

The system functions in this section are supported by DB2 CLI and defined by ODBC using vendor escape clauses.

- Arguments denoted as *exp* can be the name of a column, the result of another scalar function, or a literal.
- Arguments denoted as *value* can be a literal constant.

Table 16. System Scalar Functions

DATABASE() Returns the name of the database corresponding to the connection handle (*hdbc*). (The name of the database is also available via ***SQLGetInfo()*** by specifying the information type SQL_DATABASE_NAME.)				
DB2 for	workstation	MVS	VM/VSE	AS/400
IFNULL(*exp*, *value* **)** If *exp* is null, *value* is returned. If *exp* is not null, *exp* is returned. The possible data type(s) of *value* must be compatible with the data type of *exp*.				
DB2 for	workstation	MVS	VM/VSE	AS/400
USER() Returns the user's authorization name. (The user's authorization name is also available via ***SQLGetInfo()*** by specifying the information type SQL_USER_NAME.)				
DB2 for	workstation	MVS	VM/VSE	AS/400

Conversion Function:

The conversion function is supported by DB2 CLI and defined by ODBC using vendor escape clauses.

Each driver and data source determines which conversions are valid between the possible data types. As the driver translates the ODBC syntax into native syntax it will reject the conversions that are not supported by the data source, even if the ODBC syntax is valid.

Use the function ***SQLGetInfo()*** with the appropriate convert function masks to determine which conversions are supported by the data source.

Table 17. Conversion Function

CONVERT(*expr_value, data_type* **)** • *data_type* indicates the data type of the converted representation of *expr_value*, and can be either SQL_CHAR or SQL_DOUBLE. • *expr_value* is the value to convert. It can be of various types, depending on the conversions supported by the driver and data source. Use the function ***SQLGetInfo()*** with the appropriate convert function masks to determine which conversions are supported by the data source.				
DB2 for	workstation			

Related concepts:

- "Vendor Escape Clauses in CLI Applications" on page 199

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51

Chapter 18. Mixing Embedded SQL and DB2 CLI

Considerations for Mixing Embedded SQL and DB2 CLI

It is possible, and sometimes desirable, to use DB2 CLI in conjunction with embedded static SQL in an application. Consider the scenario where the application developer wishes to take advantage of the ease of use provided by the DB2 CLI catalog functions and maximize the portion of the application's processing where performance is critical. In order to mix the use of DB2 CLI and embedded SQL, the application must comply with the following rules:

- All connection management and transaction management must be performed completely using either DB2 CLI or embedded SQL - never a mixture of the two. Two options are available to the application:
  - it performs all connects and commits/rollbacks using DB2 CLI calls, and then calls functions written using embedded SQL;
  - or it performs all connects and commits/rollbacks using embedded SQL, and then calls functions that use DB2 CLI APIs, notably, a null connection.
- Query statement processing cannot straddle DB2 CLI and embedded SQL interfaces for the same statement. For example, the application cannot open a cursor using embedded SQL, and then call the DB2 CLI `SQLFetch()` function to retrieve row data.

Since DB2 CLI permits multiple connections, the `SQLSetConnection()` function must be called prior to executing any embedded SQL. This allows the application to explicitly specify the connection under which the embedded SQL processing is performed.

If the DB2 CLI application is multithreaded and also makes embedded SQL calls or DB2 API calls, then each thread must have a DB2 context.

Related concepts:

- "DB2 CLI versus Embedded Dynamic SQL" in the *Application Development Guide: Programming Client Applications*

Related reference:

- "SQLSetConnection Function (CLI) - Set Connection Handle" in the *CLI Guide and Reference, Volume 2*

Related samples:

- "dbmconx.c -- How to use multiple databases with embedded SQL. (CLI)"

- “dbusemx.sqc -- How to execute embedded SQL statements in CLI (CLI)”

Chapter 19. CLI/ODBC/JDBC Static Profiling

Creating Static SQL with CLI/ODBC/JDBC Static Profiling

The CLI/ODBC/JDBC Static Profiling feature enables an application's end users to replace the use of dynamic SQL with static SQL, resulting in runtime performance improvement and better security from the package-based authorization mechanism.

Restrictions:

- CLI/ODBC/JDBC Static Profiling currently targets straightforward applications. It is not meant for complex applications with many functional components and complex program logic during execution.
- When executing an application with pre-bound static SQL statements, dynamic registers that control the dynamic statement behavior will have no effect on the statements that are converted to static.
- If an application issues DDL statements for objects that are referenced in subsequent DML statements, you will find all of these statements in the capture file. The CLI/ODBC/JDBC Static Profiling Bind Tool, db2cap, will attempt to bind them. The bind attempt will be successful with DBMSs that support the VALIDATE(RUN) bind option, but it will fail with ones that do not. In this case, the application should not use Static Profiling.
- The database administrator may edit the capture file to add, change, or remove SQL statements, based on application-specific requirements.

Before running the application during the profiling session, ensure that the following conditions have been noted:

- An SQL statement must have successfully executed for it to be captured in a profiling session. In a statement matching session, unmatched dynamic statements will continue to execute as dynamic CLI/ODBC/JDBC calls.
- An SQL statement must be identical character-by-character to the one that was captured and bound to be a valid candidate for statement matching. Spaces are significant: for example, ″COL = 1″ is considered different than ″COL=1″. Use parameter markers in place of literals to improve match hits.

Be aware that there are times when not all dynamic CLI/ODBC calls can be captured and grouped into a static package. Possible reasons are:

- The application has complex control flows that make it difficult to cover all runtime conditions in a single application run.

- The application executes SET statements to change register variables. These statements are not recorded. Note that there is a limited capability in match mode to detect dynamic SET SQLID and SET SCHEMA statements, and suspend executing static statements accordingly. However, for other SET statements, subsequent SQL statements which depend on the register variables being set may not behave properly.
- The application issues DML (Data Manipulation Language) statements. Depending on application complexities and the nature of these statements, either: (1) they may not be matched, or (2) they may not execute properly at runtime.

Since dynamic and static SQL are quite different, the DBA should always verify the behaviour of the application in static match mode before making it available to end users. If CLI/ODBC/JDBC Static Profiling does not fit the type of application you are running, there are other programming methods which allow you to obtain the benefits of static SQL, such as embedded SQL and stored procedures.

Procedure:

To create static SQL statements from existing dynamic SQL statements, perform the following steps:

1. Profile the application by capturing all the dynamic SQL statements issued by the application. This process is known as running the application in static capture mode. To turn on static capture mode, set the following CLI/ODBC configuration keywords for the CLI/ODBC/JDBC data source, before running the application:
   - `STATICMODE = CAPTURE`
   - `STATICPACKAGE =` *`qualified package name`*
   - `STATICCAPFILE =` *`capture file name`*

   For example:
   ```
   [DSN1]
   STATICMODE = CAPTURE
   STATICPACKAGE = MySchema.MyPkg
   STATICCAPFILE = E:\Shared\MyApp.cpt
   ```

   Attention: For the `STATICPACKAGE` keyword, ensure that you specify a schema name (MySchema in the sample above). If a schema is not specified, the name you provide will be considered to be the container name instead of the package name, and the package name will be blank.

   The resulting static profile takes the form of a *capture file*, containing information about the SQL statements captured.

The above example file yields the following results: Data Source Name 1 (DSN1) is set to capture mode; the package named `MySchema.MyPkg` is set to be created in the current directory; and the capture file, `MyApp.cpt`, will be saved in the `E:\Shared\` directory. Until the `STATICMODE` keyword is changed to a value other than `CAPTURE`, such as `DISABLED` which is used to turn off static capture mode, each subsequent run of this application will capture SQL statements and append them to the capture file `MyApp.cpt`. Only unique SQL statements will be captured however, as duplicate executions are ignored.

2. Optional: Set the CLI/ODBC configuration keyword STATICLOGFILE to generate a CLI/ODBC/JDBC Static Profiling log file. It contains useful information to determine the state of the statement capturing process.
3. Run the application.

   Unique SQL statements will now be captured in the capture file. Duplicate statements are ignored.
4. Disable static capture mode by setting the CLI/ODBC configuration keyword STATICMODE to DISABLED, or remove the keywords set in the first step from the db2cli.ini file.
5. Bind the capture file to generate a static package by issuing the db2cap.
6. Distribute a copy of the capture file, processed with db2cap to each end user of the application. If all users reside on the same client platform, an alternative is to place a read-only copy of this capture file in a network directory accessible to all users.
7. Enable your application for dynamic-to-static SQL statement mapping, known as static match mode. Do this by setting the following CLI/ODBC configuration keywords:
   - `STATICMODE = MATCH`
   - `STATICCAPFILE =` *`capture file name`*

   For example:
   ```
   [DSN1]
   STATICMODE = MATCH
   STATICCAPFILE = E:\Shared\MyApp.cpt
   ```
8. Run the application.

Related concepts:
- "Characteristics and Reasons for Using Static SQL" in the *Application Development Guide: Programming Client Applications*
- "Capture File for CLI/ODBC/JDBC Static Profiling" on page 220

Related reference:
- "db2cap - CLI/ODBC Static Package Binding Tool" in the *Command Reference*

- “STATICCAPFILE CLI/ODBC Configuration Keyword” on page 333
- “STATICLOGFILE CLI/ODBC Configuration Keyword” on page 334
- “STATICMODE CLI/ODBC Configuration Keyword” on page 334
- “STATICPACKAGE CLI/ODBC Configuration Keyword” on page 335

Capture File for CLI/ODBC/JDBC Static Profiling

The capture file generated during static profiling is a text file. It contains the text of SQL statements and other associated information obtained in static capture mode. As well, it keeps track of a number of configurable bind options; some already contain specific values obtained from the capture run, and some are left blank, in which case the precompiler will use default values during package binding. Before binding the package(s), the DBA may want to examine the capture file and make necessary changes to these bind options using a text editor.

To help you understand how to edit SQL statements, here is the description of the fields in a statement:

Field	**Description**
SQLID	If present, indicates the SCHEMA or SQLID when the statement was captured is different from the default QUALIFIER of the package(s).
SECTNO	Section number of the static package that the statement was bound to.
ISOLATION	Isolation level for the statement. It determines which one of the five possible package the statement belongs to.
STMTTEXT	Statement string
STMTTYPE	There are 3 possible values: • SELECT_CURSOR_WITHHOLD: SELECT statement using a withhold cursor • SELECT_CURSOR_NOHOLD: SELECT statement using a nohold cursor • OTHER: non-SELECT statements
CURSOR	Cursor name declared for the SELECT statement

Field	Description
INVARnn	Description of the n-th input variable The 7 comma-separated fields refer to: 1. SQL data type 2. Length of the data. For decimal or floating point types, this is the precision. 3. For decimal or floating point types only, this is the scale. 4. TRUE if the character data is a for-bit-data type; otherwise FALSE. 5. TRUE if the variable is nullable; otherwise FALSE. 6. Column name 7. SQL_NAMED if this variable refers to a real column name; SQL_UNNAMED if the variable is a system-generate name.
OUTVARn	Description of the n-th output variable for the SELECT statement. The comma-separated fields follow the same convention as in INVARs.

Related tasks:

- "Creating Static SQL with CLI/ODBC/JDBC Static Profiling" on page 217

Chapter 20. CLI/ODBC/JDBC Trace Facility

CLI/ODBC/JDBC Trace Facility

This topic discusses the following subjects:

- "DB2 CLI and DB2 JDBC Trace Configuration"
- "DB2 CLI Trace Options and the db2cli.ini File" on page 224
- "DB2 JDBC Trace Options and the db2cli.ini File" on page 228
- "DB2 CLI Driver Trace Versus ODBC Driver Manager Trace" on page 230
- "DB2 CLI Driver, DB2 JDBC driver, and DB2 traces" on page 230
- "DB2 CLI and DB2 JDBC traces and CLI or Java Stored Procedures" on page 231

The DB2 CLI and DB2® JDBC drivers offer comprehensive tracing facilities. By default, these facilities are disabled and use no additional computing resources. When enabled, the trace facilities generate one or more text log files whenever an application accesses the appropriate driver (DB2 CLI or DB2 JDBC). These log files provide detailed information about:

- the order in which CLI or JDBC functions were called by the application
- the contents of input and output parameters passed to and received from CLI or JDBC functions
- the return codes and any error or warning messages generated by CLI or JDBC functions

DB2 CLI and DB2 JDBC trace file analysis can benefit application developers in a number of ways. First, subtle program logic and parameter initialization errors are often evident in the traces. Second, DB2 CLI and DB2 JDBC traces may suggest ways of better tuning an application or the databases it accesses. For example, if a DB2 CLI trace shows a table being queried many times on a particular set of attributes, an index corresponding to those attributes might be created on the table to improve application performance. Finally, analysis of DB2 CLI and DB2 JDBC trace files can help application developers understand how a third party application or interface is behaving.

DB2 CLI and DB2 JDBC Trace Configuration:

The configuration parameters for both DB2 CLI and DB2 JDBC traces facilities are read from the DB2 CLI configuration file db2cli.ini. By default, this file is located in the \sqllib path on the Windows® platform and the /sqllib/cfg path on UNIX® platforms. You can override the default path by setting the

DB2CLIINIPATH environment variable. On the Windows platform, an additional db2cli.ini file may be found in the user's profile (or home) directory if there are any user-defined data sources defined using the ODBC Driver Manager. This db2cli.ini file will override the default file.

To view the current db2cli.ini trace configuration parameters from the command line processor, issue the following command:

```
db2 GET CLI CFG FOR SECTION COMMON
```

There are three ways to modify the db2cli.ini file to configure the DB2 CLI and DB2 JDBC trace facilities:

- use the DB2 Configuration Assistant if it is available
- manually edit the db2cli.ini file using a text editor
- issue the UPDATE CLI CFG command from the command line processor

For example, the following command issued from the command line processor updates the db2cli.ini file and enables the JDBC tracing facility:

```
db2 UPDATE CLI CFG FOR SECTION COMMON USING jdbctrace 1
```

Notes:

1. Typically the DB2 CLI and DB2 JDBC trace configuration options are only read from the db2cli.ini configuration file at the time an application is initialized. However, a special db2cli.ini trace option, TRACEREFRESHINTERVAL, can be used to indicate an interval at which specific DB2 CLI trace options are reread from the db2cli.ini file.
2. The DB2 CLI tracing facility can also be configured dynamically by setting the SQL_ATTR_TRACE and SQL_ATTR_TRACEFILE environment attributes. These settings will override the settings contained in the db2cli.ini file.

Important: Disable the DB2 CLI and DB2 JDBC trace facilities when they are not needed. Unnecessary tracing can reduce application performance and may generate unwanted trace log files. DB2 does not delete any generated trace files and will append new trace information to any existing trace files.

DB2 CLI Trace Options and the db2cli.ini File:

When an application using the DB2 CLI driver begins execution, the driver checks for trace facility options in the [COMMON] section of the db2cli.ini file. These trace options are specific trace keywords that are set to certain values in the db2cli.ini file under the [COMMON] section.

Note: Because DB2 CLI trace keywords appear in the [COMMON] section of the db2cli.ini file, their values apply to all database connections through the DB2 CLI driver.

The DB2 CLI trace keywords that can be defined are:
- TRACE
- TRACEFILENAME
- TRACEPATHNAME
- TRACEFLUSH
- TRACEREFRESHINTERVAL
- TRACECOMM
- TRACETIMESTAMP
- TRACEPIDTID
- TRACEPIDLIST
- TRACETIME
- TRACESTMTONLY

Note: DB2 CLI trace keywords are only read from the db2cli.ini file once at application initialization time unless the TRACEREFRESHINTERVAL keyword is set. If this keyword is set, the TRACE and TRACEPIDLIST keywords are reread from the db2cli.ini file at the specified interval and applied, as appropriate, to the currently executing application.

TRACE = 0 | 1

The TRACE keyword determines whether or not any of the other DB2 CLI trace keywords have effect. If this keyword is unset or set to the default value of 0, the DB2 CLI trace facility is disabled. If this keyword is set to 1, the DB2 CLI trace facility is enabled and the other trace keywords are considered.

By itself, the TRACE keyword has little effect except to enable the DB2 CLI trace facility processing. No trace output is generated unless one of the TRACEPATHNAME or TRACEFILENAME keywords is also specified.

TRACEFILENAME = <fully_qualified_trace_file_name>

The fully qualified name of the log file to which all DB2 CLI trace information is written.

If the file does not exist, the DB2 CLI trace facility will attempt to create it. If the file already exists, new trace information for the current session, if any, will be appended to the previous contents of that file.

The TRACEFILENAME keyword option should not be used with multi-process or multithreaded applications as the trace output for all threads or processes will be written to the same log file, and the output for each thread or process will be difficult to decipher. Furthermore, semaphores are used to control access to the shared trace file which could change the behavior of multithreaded applications. There is no default DB2 CLI trace output log file name.

TRACEPATHNAME = <fully_qualified_trace_path_name>
The fully qualified path name of the directory to which all DB2 CLI trace information is written. The DB2 CLI trace facility will attempt to generate a new trace log file each time an application accessing the DB2 CLI interface is run. If the application is multithreaded, a separate trace log file will be generated for each thread. A concatenation of the application process ID and the thread sequence number is automatically used to name trace log files. There is no default path to which DB2 CLI trace output log files are written, and the path specified must exist at application execution time (the DB2 CLI driver will not create the path).

Note: If both TRACEFILENAME and TRACEPATHNAME are specified, the TRACEFILENAME keyword takes precedence and TRACEPATHNAME will be ignored.

TRACEFLUSH = 0 | <any positive integer>
The TRACEFLUSH keyword specifies how often trace information is written to the DB2 CLI trace log file. By default, TRACEFLUSH is set to 0 and each DB2 CLI trace log file is kept open until the traced application or thread terminates normally. If the application terminates abnormally, some trace information that was not written to the trace log file may be lost.

To ensure the integrity and completeness of the trace information written to the DB2 CLI trace log file, the TRACEFLUSH keyword can be specified. After n trace entries have been written to the trace log file, the DB2 CLI driver closes the file and then reopens it, appending new trace entries to the end of the file. Each file close and reopen operation incurs significant input/output overhead and can reduce performance considerably. *The smaller the value of the TRACEFLUSH keyword, the greater the impact DB2 CLI tracing has on the performance of the application*.

Setting TRACEFLUSH=1 has the most impact on performance, but will ensure that each entry is written to disk before the application continues to the next statement.

TRACEREFRESHINTERVAL = 0 | <any positive integer>
Setting TRACEREFRESHINTERVAL to a positive integer value n other than the default value of 0 causes the DB2 CLI trace facility to reread the TRACE and TRACEPIDLIST keywords from the db2cli.ini file at

the specified interval (every n seconds). The DB2 CLI trace facility then applies those keywords, as appropriate, to the trace that is currently executing.

The remaining DB2 CLI trace configuration keywords determine what information is written to the DB2 CLI trace log files.

TRACECOMM = 0 | 1
Setting TRACECOMM to the default value of 0 means no DB2 client-server communication information will be included in the DB2 CLI trace. Setting TRACECOMM to 1 causes the DB2 CLI trace to show:

- which DB2 CLI functions are processed completely on the client and which DB2 CLI functions involve communication with the server
- the number of bytes sent and received in each communication with the server
- the time spent communicating data between the client and server

TRACETIMESTAMP = 0 | 1 | 2 | 3
Setting TRACETIMESTAMP to a value other than the default of 0 means the current timestamp or absolute execution time is added to the beginning of each line of trace information as it is being written to the DB2 CLI trace log file. Setting TRACETIMESTAMP to 1 prepends the absolute execution time in seconds and milliseconds, followed by a timestamp. Setting TRACETIMESTAMP to 2 prepends the absolute execution time in seconds and milliseconds. Setting TRACETIMESTAMP to 3 prepends the timestamp.

TRACEPIDTID = 0 | 1
Setting TRACEPIDTID to the default value of 0 means process and thread ID information will not be added to each line in the DB2 CLI trace. Setting TRACEPIDTID to 1 means process and thread ID information will be included in the trace.

TRACEPIDLIST = <no value> | <pid1,pid2, pid3,...>
Setting TRACEPIDLIST to its default of no value, or leaving it unset, means all processes accessing the DB2 CLI driver interface will be traced by the DB2 CLI trace facility. Setting TRACEPIDLIST to a list of one or more comma-delimited process ID values will restrict the CLI traces generated to the processes appearing in that list.

TRACETIME = 0 | 1
Setting TRACETIME to its default value of 1, or leaving it unset, means the elapsed time between CLI function calls and returns will be calculated and included in the DB2 CLI trace. Setting TRACETIME to 0 means the elapsed time between CLI function calls and returns will not be calculated and included in the DB2 CLI trace.

TRACESTMTONLY = 0 | 1

Setting TRACESTMTONLY to its default value of 0 means trace information for all DB2 CLI function calls will be written to the DB2 CLI trace log file. Setting TRACESTMTONLY to 1 means only information related to the `SQLExecute()` and `SQLExecDirect()` function calls will be written to the log file. This trace option can be useful in determining the number of times a statement is executed in an application.

An example db2cli.ini file trace configuration using these DB2 CLI keywords and values is:

```
[COMMON]
trace=1
TraceFileName=\temp\clitrace.txt
TRACEFLUSH=1
```

Notes:

1. CLI trace keywords are NOT case sensitive. However, path and file name keyword values may be case-sensitive on some operating systems (such as UNIX).
2. If either a DB2 CLI trace keyword or its associated value in the db2cli.ini file is invalid, the DB2 CLI trace facility will ignore it and use the default value for that trace keyword instead.

DB2 JDBC Trace Options and the db2cli.ini File:

When an application using the DB2 JDBC driver begins execution, the driver also checks for trace facility options in the db2cli.ini file. As with the DB2 CLI trace options, DB2 JDBC trace options are specified as keyword/value pairs located under the [COMMON] section of the db2cli.ini file.

Note: Because DB2 JDBC trace keywords appear in the [COMMON] section of the db2cli.ini file, their values apply to all database connections through the DB2 JDBC driver.

The DB2 JDBC trace keywords that can be defined are:

- JDBCTRACE
- JDBCTRACEPATHNAME
- JDBCTRACEFLUSH

JDBCTRACE = 0 | 1

The JDBCTRACE keyword controls whether or not other DB2 JDBC tracing keywords have any effect on program execution. Setting JDBCTRACE to its default value of 0 disables the DB2 JDBC trace facility. Setting JDBCTRACE to 1 enables it.

By itself, the JDBCTRACE keyword has little effect and produces no trace output unless the JDBCTRACEPATHNAME keyword is also specified.

JDBCTRACEPATHNAME = <fully_qualified_trace_path_name>
The value of JDBCTRACEPATHNAME is the fully qualified path of the directory to which all DB2 JDBC trace information is written. The DB2 JDBC trace facility attempts to generate a new trace log file each time a JDBC application is executed using the DB2 JDBC driver. If the application is multithreaded, a separate trace log file will be generated for each thread. A concatenation of the application process ID, the thread sequence number, and a thread-identifying string are automatically used to name trace log files. There is no default path name to which DB2 JDBC trace output log files are written.

JDBCTRACEFLUSH = 0 | 1
The JDBCTRACEFLUSH keyword specifies how often trace information is written to the DB2 JDBC trace log file. By default, JDBCTRACEFLUSH is set to 0 and each DB2 JDBC trace log file is kept open until the traced application or thread terminates normally. If the application terminates abnormally, some trace information that was not written to the trace log file may be lost.

To ensure the integrity and completeness of the trace information written to the DB2 JDBC trace log file, the JDBCTRACEFLUSH keyword can be set to 1. After each trace entry has been written to the trace log file, the DB2 JDBC driver closes the file and then reopens it, appending new trace entries to the end of the file. This guarantees that no trace information will be lost.

Note: *Each DB2 JDBC log file close and reopen operation incurs significant input/output overhead and can reduce application performance considerably.*

An example db2cli.ini file trace configuration using these DB2 JDBC keywords and values is:

```
[COMMON]
jdbctrace=1
JdbcTracePathName=\temp\jdbctrace\
JDBCTRACEFLUSH=1
```

Notes:

1. JDBC trace keywords are NOT case sensitive. However, path and file name keyword values may be case-sensitive on some operating systems (such as UNIX).
2. If either a DB2 JDBC trace keyword or its associated value in the db2cli.ini file is invalid, the DB2 JDBC trace facility will ignore it and use the default value for that trace keyword instead.

3. Enabling DB2 JDBC tracing does not enable DB2 CLI tracing. Some versions of the DB2 JDBC driver depend on the DB2 CLI driver to access the database. Consequently, Java™ developers may also want to enable DB2 CLI tracing for additional information on how their applications interact with the database through the various software layers. DB2 JDBC and DB2 CLI trace options are independent of each other and can be specified together in any order under the [COMMON] section of the db2cli.ini file.

DB2 CLI Driver Trace Versus ODBC Driver Manager Trace:

It is important to understand the difference between an ODBC driver manager trace and a DB2 CLI driver trace. An ODBC driver manager trace shows the ODBC function calls made by an ODBC application to the ODBC driver manager. In contrast, a DB2 CLI driver trace shows the function calls made by the ODBC driver manager to the DB2 CLI driver *on behalf of the application*.

An ODBC driver manager might forward some function calls directly from the application to the DB2 CLI driver. However, the ODBC driver manager might also delay or avoid forwarding some function calls to the driver. The ODBC driver manager may also modify application function arguments or map application functions to other functions before forwarding the call on to the DB2 CLI driver.

Reasons for application function call intervention by the ODBC driver manager include:

- Applications written using ODBC 2.0 functions that have been deprecated in ODBC 3.0 will have the old functions mapped to new functions.
- ODBC 2.0 function arguments deprecated in ODBC 3.0 will be mapped to equivalent ODBC 3.0 arguments.
- The Microsoft® cursor library will map calls such as `SQLExtendedFetch()` to multiple calls to `SQLFetch()` and other supporting functions to achieve the same end result.
- ODBC driver manager connection pooling will usually defer `SQLDisconnect()` requests (or avoid them altogether if the connection gets reused).

For these and other reasons, application developers may find an ODBC driver manager trace to be a useful complement to the DB2 CLI driver trace.

For more information on capturing and interpreting ODBC driver manager traces, refer to the ODBC driver manager documentation. On the Windows platforms, refer to the Microsoft ODBC 3.0 Software Development Kit and Programmer’s Reference, also available online at: http://www.msdn.microsoft.com/.

DB2 CLI Driver, DB2 JDBC driver, and DB2 traces:

Internally, some versions of the DB2 JDBC driver make use of the DB2 CLI driver for database access. For example, the Java getConnection() method may be internally mapped by the DB2 JDBC driver to the DB2 CLI `SQLConnect()` function. As a result, Java developers might find a DB2 CLI trace to be a useful complement to the DB2 JDBC trace.

The DB2 CLI driver makes use of many internal and DB2 specific functions to do its work. These internal and DB2 specific function calls are logged in the DB2 trace. Application developers will not find DB2 traces useful, as they are only meant to assist IBM® Service in problem determination and resolution.

DB2 CLI and DB2 JDBC traces and CLI or Java Stored Procedures:

On all workstation platforms, the DB2 CLI and DB2 JDBC trace facilities can be used to trace DB2 CLI and DB2 JDBC stored procedures.

Most of the DB2 CLI and DB2 JDBC trace information and instructions given in earlier sections is generic and applies to both applications and stored procedures equally. However, unlike applications which are clients of a database server (and typically execute on a machine separate from the database server), stored procedures execute at the database server. Therefore, the following additional steps must be taken when tracing DB2 CLI or DB2 JDBC stored procedures:

- Ensure the trace keyword options are specified in the db2cli.ini file located at the DB2 server.
- If the TRACEREFRESHINTERVAL keyword is not set to a positive, non-zero value, ensure all keywords are configured correctly prior to database startup time (that is, when the db2start command is issued). Changing trace settings while the database server is running may have unpredictable results. For example, if the TRACEPATHNAME is changed while the server is running, then the next time a stored procedure is executed, some trace files may be written to the new path, while others are written to the original path. To ensure consistency, restart the server any time a trace keyword other than TRACE or TRACEPIDLIST is modified.

Related concepts:

- "db2cli.ini Initialization File" on page 293
- "CLI and JDBC Trace Files" on page 232

Related reference:

- "SQLSetEnvAttr Function (CLI) - Set Environment Attribute" in the *CLI Guide and Reference, Volume 2*

- “db2trc - Trace” in the *Command Reference*
- “GET CLI CONFIGURATION” in the *Command Reference*
- “UPDATE CLI CONFIGURATION” in the *Command Reference*
- “Miscellaneous variables” in the *Administration Guide: Performance*
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296

CLI and JDBC Trace Files

Applications that access the DB2® CLI and DB2 JDBC drivers can make use of the DB2 CLI and DB2 JDBC trace facilities. These utilities record all function calls made by the DB2 CLI or DB2 JDBC drivers to a log file which is useful for problem determination. This topic discusses how to access and interpret these log files generated by the tracing facilities:

- “CLI and JDBC Trace File Location”
- “CLI Trace File Interpretation” on page 234
- “JDBC Trace File Interpretation” on page 238

CLI and JDBC Trace File Location:

If the TRACEFILENAME keyword was used in the db2cli.ini file to specify a fully qualified file name, then the DB2 CLI trace log file will be in the location specified. If a relative file name was specified for the DB2 CLI trace log file name, the location of that file will depend on what the operating system considers to be the current path of the application.

Note: If the user executing the application does not have sufficient authority to write to the trace log file in the specified path, no file will be generated and no warning or error is given.

If either or both of the TRACEPATHNAME and JDBCTRACEPATHNAME keywords were used in the db2cli.ini file to specify fully qualified directories, then the DB2 CLI and DB2 JDBC trace log files will be in the location specified. If a relative directory name was specified for either or both trace directories, the operating system will determine its location based on what it considers to be the current path of the application.

Note: If the user executing the application does not have sufficient authority to write trace files in the specified path, no file will be generated and no warning or error is given. If the specified trace path does not exist, it will not be created.

The DB2 CLI and DB2 JDBC trace facilities automatically use the application’s process ID and thread sequence number to name the trace log files when the

TRACEPATHNAME and JDBCTRACEPATHNAME keywords have been set. For example, a DB2 CLI trace of an application with three threads might generate the following DB2 CLI trace log files: 100390.0, 100390.1, 100390.2.

Similarly, a DB2 JDBC trace of a Java™ application with two threads might generate the following JDBC trace log files: 7960main.trc, 7960Thread-1.trc.

Note: If the trace directory contains both old and new trace log files, file date and time stamp information can be used to locate the most recent trace files.

If no DB2 CLI or DB2 JDBC trace output files appear to have been created:

- Verify that the trace configuration keywords are set correctly in the db2cli.ini file. Issuing the `db2 GET CLI CFG FOR SECTION COMMON` command from the command line processor is a quick way to do this.
- Ensure the application is restarted after updating the db2cli.ini file. Specifically, the DB2 CLI and DB2 JDBC trace facilities are initialized during application startup. Once initialized, the DB2 JDBC trace facility cannot be reconfigured. The DB2 CLI trace facility can be reconfigured at run time but only if the TRACEREFRESHINTERVAL keyword was appropriately specified prior to application startup.

  Note: Only the TRACE and TRACEPIDLIST DB2 CLI keywords can be reconfigured at run time. *Changes made to other DB2 CLI keywords, including TRACEREFRESHINTERVAL, have no effect without an application restart*.
- If the TRACEREFRESHINTERVAL keyword was specified prior to application startup, and if the TRACE keyword was initially set to 0, ensure that enough time has elapsed for the DB2 CLI trace facility to reread the TRACE keyword value.
- If either or both the TRACEPATHNAME and JDBCTRACEPATHNAME keywords are used to specify trace directories, ensure those directories exist prior to starting the application.
- Ensure the application has write access to the specified trace log file or trace directory.
- Check the DB2CLIINIPATH environment variable. If set, the DB2 CLI and DB2 JDBC trace facilities expect the db2cli.ini file to be at the location specified by this variable.
- If the application uses ODBC to interface with the DB2 CLI driver, verify that one of the `SQLConnect()`, `SQLDriverConnect()` or `SQLBrowseConnect()` functions have been successfully called. No entries will be written to the DB2 CLI trace log files until a database connection has successfully been made.

CLI Trace File Interpretation:

DB2 CLI traces always begin with a header that identifies the process ID and thread ID of the application that generated the trace, the time the trace began, and product specific information such as the local DB2 build level and DB2 CLI driver version. For example:

```
1   [ Process: 1227, Thread: 1024 ]
2   [ Date, Time:           01-27-2002 13:46:07.535211 ]
3   [ Product:              QDB2/LINUX 7.1.0 ]
4   [ Level Identifier:     02010105 ]
5   [ CLI Driver Version:   07.01.0000 ]
6   [ Informational Tokens: "DB2 v7.1.0","n000510","" ]
```

Note: Trace examples used in this section have line numbers added to the left hand side of the trace. These line numbers have been added to aid the discussion and will *not* appear in an actual DB2 CLI trace.

Immediately following the trace header, there are usually a number of trace entries related to environment and connection handle allocation and initialization. For example:

```
7   SQLAllocEnv( phEnv=&bffff684 )
8       ---> Time elapsed - +9.200000E-004 seconds

9   SQLAllocEnv( phEnv=0:1 )
10      <--- SQL_SUCCESS   Time elapsed - +7.500000E-004 seconds

11  SQLAllocConnect( hEnv=0:1, phDbc=&bffff680 )
12      ---> Time elapsed - +2.334000E-003 seconds

13  SQLAllocConnect( phDbc=0:1 )
14      <--- SQL_SUCCESS   Time elapsed - +5.280000E-004 seconds

15  SQLSetConnectOption( hDbc=0:1, fOption=SQL_ATTR_AUTOCOMMIT, vParam=0 )
16      ---> Time elapsed - +2.301000E-003 seconds

17  SQLSetConnectOption( )
18      <--- SQL_SUCCESS   Time elapsed - +3.150000E-004 seconds

19  SQLConnect( hDbc=0:1, szDSN="SAMPLE", cbDSN=-3, szUID="", cbUID=-3,
                               szAuthStr="", cbAuthStr=-3 )
20      ---> Time elapsed - +7.000000E-005 seconds
21  ( DBMS NAME="DB2/LINUX", Version="07.01.0000", Fixpack="0x22010105" )

22  SQLConnect( )
23      <--- SQL_SUCCESS   Time elapsed - +5.209880E-001 seconds
24  ( DSN=""SAMPLE"" )

25  ( UID=" " )

26  ( PWD="*" )
```

In the above trace example, notice that there are two entries for each DB2 CLI function call (for example, lines 19-21 and 22-26 for the `SQLConnect()` function call). This is always the case in DB2 CLI traces. The first entry shows the input parameter values passed to the function call while the second entry shows the function output parameter values and return code returned to the application.

The above trace example shows that the `SQLAllocEnv()` function successfully allocated an environment handle (phEnv=0:1) at line 9. That handle was then passed to the `SQLAllocConnect()` function which successfully allocated a database connection handle (phDbc=0:1) as of line 13. Next, the `SQLSetConnectOption()` function was used to set the phDbc=0:1 connection's SQL_ATTR_AUTOCOMMIT attribute to SQL_AUTOCOMMIT_OFF (vParam=0) at line 15. Finally, `SQLConnect()` was called to connect to the target database (SAMPLE) at line 19.

Included in the input trace entry of the `SQLConnect()` function on line 21 is the build and FixPak level of the target database server. Other information that might also appear in this trace entry includes input connection string keywords and the code pages of the client and server. For example, suppose the following information also appeared in the `SQLConnect()` trace entry:

```
( Application Codepage=819, Database  Codepage=819,
  Char Send/Recv Codepage=819, Graphic Send/Recv Codepage=819,
  Application Char Codepage=819, Application Graphic Codepage=819 )
```

This would mean the application and the database server were using the same code page (819).

The return trace entry of the `SQLConnect()` function also contains important connection information (lines 24-26 in the above example trace). Additional information that might be displayed in the return entry includes any PATCH1 or PATCH2 keyword values that apply to the connection. For example, if PATCH2=27,28 was specified in the db2cli.ini file under the COMMON section, the following line should also appear in the `SQLConnect()` return entry:

```
( PATCH2="27,28" )
```

Following the environment and connection related trace entries are the statement related trace entries. For example:

```
27  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
28      ---> Time elapsed - +1.868000E-003 seconds

29  SQLAllocStmt( phStmt=1:1 )
30      <--- SQL_SUCCESS   Time elapsed - +6.890000E-004 seconds

31  SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                                  VARCHAR(10))", cbSqlStr=-3 )
```

```
32      ---> Time elapsed - +2.863000E-003 seconds
33  ( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

34  SQLExecDirect( )
35      <--- SQL_SUCCESS   Time elapsed - +2.387800E-002 seconds
```

In the above trace example, the database connection handle (phDbc=0:1) was used to allocate a statement handle (phStmt=1:1) at line 29. An unprepared SQL statement was then executed on that statement handle at line 31. If the TRACECOMM=1 keyword had been set in the db2cli.ini file, the `SQLExecDirect()` function call trace entries would have shown additional client-server communication information as follows:

```
SQLExecDirect( hStmt=1:1, pszSqlStr="CREATE TABLE GREETING (MSG
                         VARCHAR(10))", cbSqlStr=-3 )
    ---> Time elapsed - +2.876000E-003 seconds
( StmtOut="CREATE TABLE GREETING (MSG VARCHAR(10))" )

    sqlccsend( ulBytes - 232 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.150000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 163 ) - rc - 0, time elapsed - +2.243800E-002

SQLExecDirect( )
    <--- SQL_SUCCESS   Time elapsed - +2.384900E-002 seconds
```

Notice the additional sqlccsend() and sqlccrecv() function call information in this trace entry. The sqlccsend() call information reveals how much data was sent from the client to the server, how long the transmission took, and the success of that transmission (0 = SQL_SUCCESS). The sqlccrecv() call information then reveals how long the client waited for a response from the server and the amount of data included in the response.

Often, multiple statement handles will appear in the DB2 CLI trace. By paying close attention to the statement handle identifier, one can easily follow the execution path of a statement handle independent of all other statement handles appearing in the trace.

Statement execution paths appearing in the DB2 CLI trace are usually more complicated than the example shown above. For example:

```
36  SQLAllocStmt( hDbc=0:1, phStmt=&bffff684 )
37      ---> Time elapsed - +1.532000E-003 seconds

38  SQLAllocStmt( phStmt=1:2 )
39      <--- SQL_SUCCESS   Time elapsed - +6.820000E-004 seconds

40  SQLPrepare( hStmt=1:2, pszSqlStr="INSERT INTO GREETING VALUES ( ? )",
                cbSqlStr=-3 )
41      ---> Time elapsed - +2.733000E-003 seconds
42  ( StmtOut="INSERT INTO GREETING VALUES ( ? )" )
```

```
43  SQLPrepare( )
44      <--- SQL_SUCCESS   Time elapsed - +9.150000E-004 seconds

45  SQLBindParameter( hStmt=1:2, iPar=1, fParamType=SQL_PARAM_INPUT,
                      fCType=SQL_C_CHAR, fSQLType=SQL_CHAR, cbColDef=14,
                      ibScale=0, rgbValue=&080eca70, cbValueMax=15,
                      pcbValue=&080eca4c )
46      ---> Time elapsed - +4.091000E-003 seconds

47  SQLBindParameter( )
48      <--- SQL_SUCCESS   Time elapsed - +6.780000E-004 seconds

49  SQLExecute( hStmt=1:2 )
50      ---> Time elapsed - +1.337000E-003 seconds
51  ( iPar=1, fCType=SQL_C_CHAR, rgbValue="Hello World!!!", pcbValue=14,
      piIndicatorPtr=14 )

52  SQLExecute( )
53      <--- SQL_ERROR   Time elapsed - +5.951000E-003 seconds
```

In the above trace example, the database connection handle (phDbc=0:1) was used to allocate a second statement handle (phStmt=1:2) at line 38. An SQL statement with one parameter marker was then prepared on that statement handle at line 40. Next, an input parameter (iPar=1) of the appropriate SQL type (SQL_CHAR) was bound to the parameter marker at line 45. Finally, the statement was executed at line 49. Notice that both the contents and length of the input parameter (rgbValue="Hello World!!!", pcbValue=14) are displayed in the trace on line 51.

The `SQLExecute()` function fails at line 52. If the application calls a diagnostic DB2 CLI function like `SQLError()` to diagnose the cause of the failure, then that cause will appear in the trace. For example:

```
54  SQLError( hEnv=0:1, hDbc=0:1, hStmt=1:2, pszSqlState=&bffff680,
              pfNativeError=&bfffee78, pszErrorMsg=&bffff280,
              cbErrorMsgMax=1024, pcbErrorMsg=&bfffee76 )
55      ---> Time elapsed - +1.512000E-003 seconds

56  SQLError( pszSqlState="22001", pfNativeError=-302, pszErrorMsg="[IBM][CLI
      Driver][DB2/LINUX] SQL0302N  The value of a host variable in the EXECUTE
      or OPEN statement is too large for its corresponding use.
      SQLSTATE=22001", pcbErrorMsg=157 )
57      <--- SQL_SUCCESS   Time elapsed - +8.060000E-004 seconds
```

The error message returned at line 56 contains the DB2 native error code that was generated (SQL0302N), the sqlstate that corresponds to that code (SQLSTATE=22001) and a brief description of the error. In this example, the source of the error is evident: on line 49, the application is trying to insert a string with 14 characters into a column defined as VARCHAR(10) on line 31.

If the application does not respond to a DB2 CLI function warning or error return code by calling a diagnostic function like `SQLError()`, the warning or error message should still be written to the DB2 CLI trace. However, the location of that message in the trace may not be close to where the error actually occurred. Furthermore, the trace will indicate that the error or warning message was not retrieved by the application. For example, if not retrieved, the error message in the above example might not appear until a later, seemingly unrelated DB2 CLI function call as follows:

```
SQLDisconnect( hDbc=0:1 )
    –––> Time elapsed - +1.501000E-003 seconds
    sqlccsend( ulBytes - 72 )
    sqlccsend( Handle - 1084869448 )
    sqlccsend( ) - rc - 0, time elapsed - +1.080000E-004
    sqlccrecv( )
    sqlccrecv( ulBytes - 27 ) - rc - 0, time elapsed - +1.717950E-001
( Unretrieved error message="SQL0302N  The value of a host variable in the
  EXECUTE or OPEN statement is too large for its corresponding use.
  SQLSTATE=22001" )

SQLDisconnect( )
    <––– SQL_SUCCESS   Time elapsed - +1.734130E-001 seconds
```

The final part of a DB2 CLI trace should show the application releasing the database connection and environment handles that it allocated earlier in the trace. For example:

```
SQLTransact( hEnv=0:1, hDbc=0:1, fType=SQL_ROLLBACK )
    –––> Time elapsed - +6.085000E-003 seconds
( ROLLBACK=0 )

SQLTransact( )
    <––– SQL_SUCCESS   Time elapsed - +2.220750E-001 seconds

SQLDisconnect( hDbc=0:1 )
    –––> Time elapsed - +1.511000E-003 seconds

SQLDisconnect( )
    <––– SQL_SUCCESS   Time elapsed - +1.531340E-001 seconds

SQLFreeConnect( hDbc=0:1 )
    –––> Time elapsed - +2.389000E-003 seconds

SQLFreeConnect( )
    <––– SQL_SUCCESS   Time elapsed - +3.140000E-004 seconds

SQLFreeEnv( hEnv=0:1 )
    –––> Time elapsed - +1.129000E-003 seconds

SQLFreeEnv( )
    <––– SQL_SUCCESS   Time elapsed - +2.870000E-004 seconds
```

JDBC Trace File Interpretation:

DB2 JDBC traces always begin with a header that lists important system information such as key environment variable settings, the JDK or JRE level, the DB2 JDBC driver level, and the DB2 build level. For example:

```
1   ========================================================
2   |    Trace beginning on 2002-1-28 7:21:0.19
3   ========================================================

4   System Properties:
5   ------------------
6   user.language = en
7   java.home = c:\Program Files\SQLLIB\java\jdk\bin\..
8   java.vendor.url.bug =
9   awt.toolkit = sun.awt.windows.WToolkit
10  file.encoding.pkg = sun.io
11  java.version = 1.1.8
12  file.separator = \
13  line.separator =
14  user.region = US
15  file.encoding = Cp1252
16  java.compiler = ibmjitc
17  java.vendor = IBM® Corporation
18  user.timezone = EST
19  user.name = db2user
20  os.arch = x86
21  java.fullversion = JDK 1.1.8 IBM build n118p-19991124 (JIT ibmjitc
                       V3.5-IBMJDK1.1-19991124)
22  os.name = Windows® NT
23  java.vendor.url = http://www.ibm.com/
24  user.dir = c:\Program Files\SQLLIB\samples\java
25  java.class.path =
         .:C:\Program Files\SQLLIB\lib;C:\Program Files\SQLLIB\java;
         C:\Program Files\SQLLIB\java\jdk\bin\
26  java.class.version = 45.3
27  os.version = 5.0
28  path.separator = ;
29  user.home = C:\home\db2user
30  ----------------------------------------
```

Note: Trace examples used in this section have line numbers added to the left hand side of the trace. These line numbers have been added to aid the discussion and will *not* appear in an actual DB2 JDBC trace.

Immediately following the trace header, one usually finds a number of trace entries related to initialization of the JDBC environment and database connection establishment. For example:

```
31  jdbc.app.DB2Driver -> DB2Driver() (2002-1-28 7:21:0.29)
32  | Loaded db2jdbc from java.library.path
33  jdbc.app.DB2Driver <- DB2Driver() [Time Elapsed = 0.01]

34  DB2Driver - connect(jdbc:db2:sample)

35  jdbc.app.DB2ConnectionTrace -> connect( sample, info, db2driver, 0, false )
```

```
                   (2002-1-28 7:21:0.59)
36  | 10: connectionHandle = 1
37  jdbc.app.DB2ConnectionTrace <— connect() [Time Elapsed = 0.16]

38  jdbc.app.DB2ConnectionTrace —> DB2Connection (2002-1-28 7:21:0.219)
39  | source = sample
40  | Connection handle = 1
41  jdbc.app.DB2ConnectionTrace <— DB2Connection
```

In the above trace example, a request to load the DB2 JDBC driver was made on line 31. This request returned successfully as reported on line 33.

The DB2 JDBC trace facility uses specific Java classes to capture the trace information. In the above trace example, one of those trace classes, DB2ConnectionTrace, has generated two trace entries numbered 35-37 and 38-41.

Line 35 shows the connect() method being invoked and the input parameters to that method call. Line 37 shows that the connect() method call has returned successfully while line 36 shows the output parameter of that call (Connection handle = 1).

Following the connection related entries, one usually finds statement related entries in the JDBC trace. For example:

```
42  jdbc.app.DB2ConnectionTrace —> createStatement() (2002-1-28 7:21:0.219)
43  | Connection handle = 1
44  | jdbc.app.DB2StatementTrace —> DB2Statement( con, 1003, 1007 )
          (2002-1-28 7:21:0.229)
45  | jdbc.app.DB2StatementTrace <— DB2Statement() [Time Elapsed = 0.0]
46  | jdbc.app.DB2StatementTrace —> DB2Statement (2002-1-28 7:21:0.229)
47  | | Statement handle = 1:1
48  | jdbc.app.DB2StatementTrace <— DB2Statement
49  jdbc.app.DB2ConnectionTrace <— createStatement - Time Elapsed = 0.01

50  jdbc.app.DB2StatementTrace —> executeQuery(SELECT * FROM EMPLOYEE WHERE
                                 empno = 000010) (2002-1-28 7:21:0.269)
51  | Statement handle = 1:1
52  | jdbc.app.DB2StatementTrace —> execute2( SELECT * FROM EMPLOYEE WHERE
                                   empno = 000010 ) (2002-1-28 7:21:0.269)
52  | | jdbc.DB2Exception —> DB2Exception() (2002-1-28 7:21:0.729)
53  | | | 10: SQLError = [IBM][CLI Driver][DB2/NT] SQL0401N  The data types of
                the operands for the operation "=" are not compatible.
                SQLSTATE=42818
54  | | |     SQLState = 42818
55  | | |     SQLNativeCode = -401
56  | | |     LineNumber = 0
57  | | |     SQLerrmc = =
58  | | jdbc.DB2Exception <— DB2Exception() [Time Elapsed = 0.0]
59  | jdbc.app.DB2StatementTrace <— executeQuery - Time Elapsed = 0.0
```

On line 42 and 43, the DB2ConnectionTrace class reported that the JDBC createStatement() method had been called with connection handle 1. Within that method, the internal method DB2Statement() was called as reported by another DB2 JDBC trace facility class, DB2StatementTrace. Notice that this internal method call appears ′nested′ in the trace entry. Lines 47-49 show that the methods returned successfully and that statement handle 1:1 was allocated.

On line 50, an SQL query method call is made on statement 1:1, but the call generates an exception at line 52. The error message is reported on line 53 and contains the DB2 native error code that was generated (SQL0401N), the sqlstate that corresponds to that code (SQLSTATE=42818) and a brief description of the error. In this example, the error results because the EMPLOYEE.EMPNO column is defined as CHAR(6) and not an integer value as assumed in the query.

Related concepts:
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:
- "Miscellaneous variables" in the *Administration Guide: Performance*
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACECOMM CLI/ODBC Configuration Keyword" on page 340
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343
- "TRACEPIDLIST CLI/ODBC Configuration Keyword" on page 344
- "TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword" on page 346

Chapter 21. CLI Bind Files and Package Names

DB2 CLI Bind Files and Package Names

The CLI/ODBC driver will automatically bind changed or unbound CLI packages when required, provided the user has the appropriate privilege or authorization. This typically occurs the first time a database is accessed once it has been created, or the first time that it has been accessed by a new DB2 client version. The administrator may want to supervise the autobind process (typically the first execution of a CLI program) or explicitly bind the required files.

Table 18. DB2 CLI Bind Files and Package Names

Bind File Name	Package Name	Needed by DB2 Universal Database	Needed by host servers	Description
db2clish.bnd	SYSSHxyy	Yes	Yes	Dynamic Placeholders - Small Package WITH HOLD
db2clisn.bnd	SYSSNxyy	Yes	Yes	Dynamic Placeholders - Small Package NO WITH HOLD
db2clibh.bnd	SYSLHxyy	Yes	Yes	Dynamic Placeholders - Large Package WITH HOLD
db2clibn.bnd	SYSLNxyy	Yes	Yes	Dynamic Placeholders - Large Package NO WITH HOLD
db2clist.bnd	SYSSTAT	Yes	Yes	Common Static CLI Functions
db2schema.bnd	SYSSCwww	Yes	Yes	Catalog Function Support

Table 18. DB2 CLI Bind Files and Package Names (continued)

Bind File Name	Package Name	Needed by DB2 Universal Database	Needed by host servers	Description
db2cliws.bnd	SQLL65zz	Server Version 2 to 7	No	DB2 for Intel/UNIX Catalog Function Support
db2cliv2.bnd	SQLL95zz	Server Version 2 to 7	No	Common Static CLI Functions

Note:

- Where ’www’ is the package iteration, 000 through ZZZ. That is, each digit is a base-36 number: 000, 001, ..., 009, 00A, 00B, ..., 00Z, 010, ..., 01Z, 020, ...
- Where ’x’ is the isolation level: 0=NC, 1=UR, 2=CS, 4=RS, 8=RR
- Where ’yy’ is the package iteration 00 through FF
- Where ’zz’ is unique for each platform.

For example, for the dynamic packages:

- SYSSN100 A small package (64 sections) where all cursor declarations are for non-held cursors. Bound with isolation level UR. This is the first iteration of that package.
- SYSLH401 A large package (384 sections) where all cursor declarations are for held cursors. Bound with isolation level RS. This is the second iteration of that package.

Previous versions of DB2 servers do not need all of the bind files and will therefore return errors at bind time. Use the bind option SQLERROR(CONTINUE) so that the same package can be bound on all platforms and errors will be ignored for any statements not supported there.

The db2cli.lst file contains the names of the required bind files for DB2 CLI to connect to DB2 Version 8 servers (db2clish.bnd, db2clisn.bnd, db2clibh.bnd, db2clibn.bnd, db2clist.bnd, and db2schema.bnd).

For host and iSeries servers use one of ddcsvm.lst, ddcsmvs.lst, ddcsvse.lst, or ddcs400.lst bind list files.

Related tasks:

- “Setting Up the CLI Environment” on page 247

Part 3. CLI Environment and Application Building

Chapter 22. CLI Environmental Setup

Before you can run a CLI application, you must set up the CLI environment. This chapter describes how to set up the CLI or ODBC environment on the UNIX and Windows platforms.

Setting Up the CLI Environment

Runtime support for DB2 CLI applications is contained in all DB2 clients. Support for building and running DB2 CLI applications is contained in the DB2 Application Development (DB2 AD) Client. This section describes the general setup required for DB2 CLI runtime support.

The CLI/ODBC driver will autobind on the first connection to the database, provided the user has the appropriate privilege or authorization. The administrator may want to perform the first connect or explicitly bind the required files.

Prerequisites:

Before you set up your CLI environment, ensure you have set up the application development environment.

Procedure:

In order for a DB2 CLI application to successfully access a DB2 database:

1. Ensure the DB2 CLI/ODBC driver was installed during the DB2 client install.
2. Catalog the DB2 database and node if the database is being accessed from a remote client.

   On the Windows platform, you can use the CLI/ODBC Settings GUI to catalog the DB2 database.
3. Optional: Explicitly bind the DB2 CLI/ODBC bind files to the database with the command:

   ```
   db2 bind ~/sqllib/bnd/@db2cli.lst blocking all sqlerror continue \
            messages cli.msg grant public
   ```

   On the Windows platform, you can use the CLI/ODBC Settings GUI to bind the DB2 CLI/ODBC bind files to the database.

4. Optional: Change the DB2 CLI/ODBC configuration keywords by editing the `db2cli.ini file`, located in the `sqllib` directory on Windows, and in the `sqllib/cfg` directory on UNIX platforms.

   On the Windows platform, you can use the CLI/ODBC Settings GUI to set the DB2 CLI/ODBC configuration keywords.

Once you have completed the above steps, proceed to setting up your Windows CLI environment, or setting up your UNIX ODBC environment if you are running ODBC applications on UNIX.

Related concepts:

- "Initialization and Termination in CLI Overview" on page 22

Related tasks:

- "Initializing CLI Applications" on page 24
- "Setting Up the UNIX ODBC Environment" on page 248
- "Setting Up the Windows CLI Environment" on page 250

Related reference:

- "BIND" in the *Command Reference*
- "CATALOG DATABASE" in the *Command Reference*

Setting Up the UNIX ODBC Environment

This topic explains how to set up UNIX client access to DB2 for ODBC applications. (If your application is a DB2 CLI application, your CLI environmental setup will be complete once the task in the Prerequisites section is performed.)

Prerequisites:

Before setting up the UNIX ODBC environment, ensure you have set up the CLI environment.

Procedure:

For ODBC applications on UNIX that need to access a DB2 database, follow the steps described below.

1. Ensure that an ODBC Driver Manager is installed and that each user that will use ODBC has access to it. DB2 does not install an ODBC Driver Manager, so you must use the ODBC Driver Manager that was supplied with your ODBC client application or ODBC SDK in order to access DB2 data using that application.

2. Set up `.odbc.ini`, the end-user's data source configuration. Each user ID has a separate copy of this file in their home directory. Note that the file starts with a dot. Although necessary files are usually updated automatically by the tools on most platforms, users of ODBC on UNIX platforms will have to edit them manually.

   Using an ASCII editor, update the file to reflect the appropriate data source configuration information. To register a DB2 database as an ODBC data source there must be one stanza (section) for each DB2 database.

   The .odbc.ini file must contain the following lines:

   - in the `[ODBC Data Source]` stanza:

     ```
     SAMPLE=IBM DB2 ODBC DRIVER
     ```

     which indicates that there is a data source called SAMPLE that used the IBM DB2 ODBC DRIVER;
   - in the `[SAMPLE]` stanza:

     on AIX, for example,

     ```
     [SAMPLE]
     Driver=/u/thisuser/sqllib/lib/libdb2.a
     Description=Sample DB2 ODBC Database
     ```

     on Solaris, for example,

     ```
     [SAMPLE]
     Driver=/u/thisuser/sqllib/lib/libdb2.so
     Description=Sample DB2 ODBC Database
     ```

     which indicates that the SAMPLE database is part of the DB2 instance located in the directory `/u/thisuser`.
3. Ensure that the application execution environment has reference to the ODBC Driver Manager by including `libodbc.a` (for AIX) or `libodbc.so` (for UNIX) in the LIBPATH (for AIX) or LD_LIBRARY_PATH (for UNIX) environment variables.
4. Enable a system-wide `.odbc.ini` file to be used by setting the ODBCINI environment variable to the fully qualified pathname of the .ini file. Some ODBC driver managers support this feature which allows for centralized control. The following examples show how to set ODBCINI:

   in the C shell,

   ```
   setenv ODBCINI /opt/odbc/system_odbc.ini
   ```

   in the Bourne or Korn shell,

   ```
   ODBCINI=/opt/odbc/system_odbc.ini;export ODBCINI
   ```
5. Once the `.odbc.ini` file is set up, you can run your ODBC application and access DB2 databases. Refer to the documentation that comes with your ODBC application for additional help and information.

Related concepts:
- "Comparison of DB2 CLI and Microsoft ODBC" on page 11
- "Initialization and Termination in CLI Overview" on page 22

Related tasks:
- "Initializing CLI Applications" on page 24
- "Setting Up the CLI Environment" on page 247
- "Building CLI Applications on UNIX" on page 253
- "Building CLI Routines on UNIX" on page 255

Setting Up the Windows CLI Environment

This task tells you how to perform Windows client access to DB2 using CLI or ODBC.

Prerequisites:

Before setting up the Windows CLI environment, ensure that the CLI environment has been set up.

Restrictions:

When using the Configuration Assistant on Windows 64-bit platforms, ODBC Data Sources can be configured *only* for 64-bit applications. ODBC Data Sources for 32-bit applications need to be configured using the Microsoft 32-bit ODBC Data Source Administrator (32-bit odbcad32.exe) that is included with the Windows 64-bit operating system.

Procedure:

Before DB2 CLI and ODBC applications can successfully access a DB2 database from a Windows client, perform the following steps on the client system:

1. Verify that the Microsoft ODBC Driver Manager and the DB2 CLI/ODBC driver are installed. On Windows operating systems they are both installed with DB2 unless the ODBC component is manually unselected during the install. DB2 will not overwite a newer version of the Microsoft ODBC Driver Manager if one is found. To verify that they both exist on the machine:
   a. Start the Microsoft ODBC Data Sources icon in the Control Panel, or run the `odbcad32.exe` command from the command line.
   b. Click on the ″Drivers″ tab.
   c. Verify that `IBM DB2 ODBC DRIVER` is shown in the list.

If either the Microsoft ODBC Driver Manager or the IBM DB2 CLI/ODBC driver is not installed, then rerun the DB2 install and select the ODBC component on Windows operating systems.

Note: The latest version of the Microsoft ODBC Driver Manager is included as part of the Microsoft Data Access Components (MDAC) and is available for download from http://www.microsoft.com/data/.

2. Register the DB2 database with the ODBC driver manager as a data source. On Windows operating systems you can make the data source available to all users of the system (a system data source), or only the current user (a user data source). Use either of these methods to add the data source:
   - Using the Configuration Assistant:
     a. Select the DB2 database alias that you want to add as a data source.
     b. Click on the ″Properties″ push button. The Database Properties window opens.
     c. Select the ″Register this database for ODBC″ check box.
     d. Use the radio buttons to add the data source as either a user, system, or file data source.
   - Using the Microsoft ODBC Administration tool, which you can access from the icon in the Control Panel or by running `odbcad32.exe` from the command line:
     a. The list of user data sources appears by default. If you want to add a system data source click on the ″System DSN″ button, or the ″System DSN″ tab (depending on the platform).
     b. Click on the ″Add″ push button.
     c. Double-click on the IBM DB2 ODBC Driver in the list.
     d. Select the DB2 database to add and click on OK.
   - Use the `CATALOG` command to register the DB2 database with the ODBC driver manager as a data source:
     ```
     CATALOG [ user | system ] ODBC DATA SOURCE
     ```
     Using this command, an administrator could create a command line processor script to register the required databases. This script could then be run on all machines that require access to DB2 databases through ODBC.
3. Optional: Configure the DB2 CLI/ODBC driver using the Configuration Assistant:
   a. Select the DB2 database alias you want to configure.
   b. Click on the ″Properties″ push button. The Database Properties window opens.

   c. Click on the ″Settings″ push button. The CLI/ODBC Settings window opens.
   d. Click on the ″Advanced″ push button. You can set the configuration keywords in the window that opens. These keywords are associated with the database alias name, and affect all DB2 CLI/ODBC applications that access the database.
4. If you have installed ODBC access (as described above), you can now access DB2 data using ODBC applications.

Related concepts:

- "db2cli.ini Initialization File" on page 293
- "Initialization and Termination in CLI Overview" on page 22

Related tasks:

- "Initializing CLI Applications" on page 24
- "Setting Up the CLI Environment" on page 247
- "Building CLI Applications on Windows" on page 279
- "Building CLI Routines on Windows" on page 281

Chapter 23. Building CLI Applications

UNIX

The following sections describe how to build CLI applications and routines on the UNIX platform, and provide sample build scripts for each platform with descriptions of their compile and link options.

Building CLI Applications on UNIX

DB2 provides build scripts for compiling and linking CLI programs. These are located in the `sqllib/samples/cli` directory, along with sample programs that can be built with these files.

The script file `bldapp` contains the commands to build a DB2 CLI application. It takes up to four parameters, represented inside the script file by the variables `$1`, `$2`, `$3`, and `$4`.

The parameter, `$1`, specifies the name of your source file. This is the only required parameter, and the only one needed for CLI applications that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: the second parameter, `$2`, specifies the name of the database to which you want to connect; the third parameter, `$3`, specifies the user ID for the database, and `$4` specifies the password.

If the program contains embedded SQL, indicated by the `.sqc` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

Procedure:

The following examples show you how to build and run CLI applications.

To build the sample program `tbinfo` from the source file `tbinfo.c`, enter:

```
bldapp tbinfo
```

The result is an executable file, `tbinfo`. You can run the executable file by entering the executable name:

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp dbusemx database userid password
   ```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```

3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

Related tasks:
- "Setting Up the Application Development Environment" in the *Application Development Guide: Building and Running Applications*
- "Setting Up the UNIX Application Development Environment" in the *Application Development Guide: Building and Running Applications*
- "Building CLI Routines on UNIX" on page 255

Related reference:
- "AIX CLI Application Compile and Link Options" on page 258
- "HP-UX CLI Application Compile and Link Options" on page 265
- "Linux CLI Application Compile and Link Options" on page 270
- "Solaris CLI Application Compile and Link Options" on page 274

Related samples:
- "bldapp -- Builds AIX CLI applications"
- "dbusemx.sqc -- How to execute embedded SQL statements in CLI (CLI)"
- "tbinfo.c -- How to get information about tables from the system catalog tables (CLI)"

Building CLI Routines on UNIX

DB2 provides build scripts for compiling and linking CLI programs. These are located in the `sqllib/samples/cli` directory, along with sample programs that can be built with these files.

The script file `bldrtn` contains the commands to build DB2 CLI routines (stored procedures and user-defined functions). `bldrtn` creates a shared library on the server. It takes a parameter for the source file name, represented inside the script file by the variable $1.

Procedure:

To build the sample program `spserver` from the source file `spserver.c`:
1. Enter the build script name and program name:
   ```
   bldrtn spserver
   ```
   The script file copies the shared library to the `sqllib/function` directory.
2. Next, catalog the routines by running the `spcat` script on the server:
   ```
   spcat
   ```

This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by calling `spcreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spcreate.db2` scripts individually.

3. Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the shared library, `spserver`, you can build the CLI client application, `spclient`, that calls the routines within the shared library.

The client application can be built like any other CLI client application by using the script file, `bldapp`.

To access the shared library, run the sample client application by entering:

`spclient` *database userid password*

where

database
: Is the name of the database to which you want to connect. The name could be `sample`, or its alias, or another database name.

userid
: Is a valid user ID.

password
: Is a valid password.

The client application accesses the shared library, `spserver`, and executes the routines on the server database. The output is returned to the client application.

Related tasks:

- "Setting Up the UNIX Application Development Environment" in the *Application Development Guide: Building and Running Applications*
- "Building CLI Applications on UNIX" on page 253

Related reference:

- "AIX CLI Routine Compile and Link Options" on page 262
- "HP-UX CLI Routine Compile and Link Options" on page 268
- "Linux CLI Routine Compile and Link Options" on page 272
- "Solaris CLI Routine Compile and Link Options" on page 277

Related samples:

- “bldrtn -- Builds AIX CLI routines (stored procedures and UDFs)”
- “spclient.c -- Call various stored procedures (CLI)”
- “spserver.c -- Definition of various types of stored procedures (CLI)”
- “spcat -- To catalog C stored procedures on UNIX”
- “spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc ”
- “spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc ”

AIX

Build Script for AIX Applications

The following is the `bldapp` script for building CLI applications on AIX:

```
#! /bin/sh

# SCRIPT: bldapp
# Builds AIX CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
xlc $EXTRA_CFLAG -I$DB2PATH/include -c utilcli.c

# Compile the program.
xlc $EXTRA_CFLAG -I$DB2PATH/include -c $1.c

# Link the program.
xlc $EXTRA_CFLAG -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

AIX CLI Application Compile and Link Options

These compile and link options are recommended by DB2 for building CLI applications with the AIX IBM C compiler. They are demonstrated in the `sqllib/samples/cli/bldapp` build script.

Compile and Link Options for bldapp	
Compile Options:	
`xlc`	The IBM C compiler.
`$EXTRA_CFLAG`	Contains ″-q64″ value if ′BUILD_64BIT=true′ is uncommented; otherwise, it contains no value.
`-I$DB2PATH/include`	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
`-c`	Perform compile only; no link. This script has separate compile and link steps.
Link Options:	
`xlc`	Use the compiler as a front end for the linker.
`$EXTRA_CFLAG`	Contains ″-q64″ value if ′BUILD_64BIT=true′ is uncommented; otherwise, it contains no value.
`-o $1`	Specify the executable program.
`$1.o`	Specify the object file.
`utilcli.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
`-ldb2`	Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.	

Related tasks:

- "Building CLI Applications on UNIX" on page 253
- "Building CLI Applications with Configuration Files" on page 259

Related reference:

- "AIX CLI Routine Compile and Link Options" on page 262

Related samples:

- "bldapp -- Builds AIX CLI applications"

CLI Applications and Configuration Files on AIX

The following section describes how to build CLI applications with configuration files. This method of building CLI applications is only available on AIX using the VisualAge C++ compiler.

Building CLI Applications with Configuration Files: The configuration file, `cli.icc`, in `sqllib/samples/cli` allows you to build DB2 CLI programs.

Procedure:

To use the configuration file to build the DB2 CLI sample program `tbinfo` from the source file `tbinfo.c`, do the following:

1. Set the CLI environment variable:
   ```
   export CLI=tbinfo
   ```
2. If you have a `cli.ics` file in your working directory, produced by building a different program with the `cli.icc` file, delete the `cli.ics` file with this command:
   ```
   rm cli.ics
   ```
   An existing `cli.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld cli.icc
   ```
   Note: The `vacbld` command is provided by VisualAge C++.

The result is an executable file, `tbinfo`. You can run the program by entering the executable name:

```
tbinfo
```

Building and Running Embedded SQL Applications

You use the configuration file after the program is precompiled with the `embprep` file. The `embprep` file precompiles the source file and binds the program to the database. You use the `cli.icc` configuration file to compile the precompiled file.

There are three ways to precompile the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   embprep dbusemx
   ```

2. If connecting to another database on the same instance, also enter the database name:
   ```
   embprep dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   embprep dbusemx database userid password
   ```

The result is a precompiled C file, `dbusemx.c`.

After it is precompiled, the C file can be compiled with the `cli.icc` file as follows:
1. Set the CLI environment variable to the program name by entering:
   ```
   export CLI=dbusemx
   ```
2. If you have a `cli.ics` file in your working directory, produced by building a different program with the `cli.icc` or `cliapi.icc` file, delete the `cli.ics` file with this command:
   ```
   rm cli.ics
   ```

   An existing `cli.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld cli.icc
   ```

There are three ways to run this embedded SQL application:
1. If accessing the `sample` database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

Related tasks:
- "Building CLI Stored Procedures with Configuration Files" on page 263
- "Building C++ Embedded SQL Applications with Configuration Files" in the *Application Development Guide: Building and Running Applications*
- "Building C++ DB2 API Applications with Configuration Files" in the *Application Development Guide: Building and Running Applications*
- "Building C++ Stored Procedures with Configuration Files" in the *Application Development Guide: Building and Running Applications*

- "Building C++ User-defined Functions with Configuration Files" in the *Application Development Guide: Building and Running Applications*
- "Building VisualAge C++ Programs with Configuration Files" in the *Application Development Guide: Building and Running Applications*

Related samples:

- "dbusemx.sqc -- How to execute embedded SQL statements in CLI (CLI)"
- "tbinfo.c -- How to get information about tables from the system catalog tables (CLI)"

Build Script for AIX Routines

The following is the `bldrtn` script for building CLI routines on AIX:

```
#! /bin/sh

# SCRIPT: bldrtn
# Builds AIX CLI routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG=-q64
else
  EXTRA_CFLAG=
fi

# Compile the error-checking utility.
xlc_r $EXTRA_CFLAG -I$DB2PATH/include -c utilcli.c

# Compile the program.
xlc_r $EXTRA_CFLAG -I$DB2PATH/include -c $1.c

# Link the program.
xlc_r $EXTRA_CFLAG -qmkshrobj -o $1 $1.o utilcli.o -L$DB2PATH/lib \
  -ldb2 -bE:$1.exp

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

AIX CLI Routine Compile and Link Options

These compile and link options are recommended by DB2 for building CLI routines (stored procedures and user-defined functions) with the AIX IBM C compiler. They are demonstrated in the `sqllib/samples/cli/bldrtn` build script.

Compile and Link Options for bldrtn	
Compile Options:	
`xlc_r`	Use the multi-threaded version of the IBM C compiler, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
`$EXTRA_CFLAG`	Contains ″-q64″ value if ’BUILD_64BIT=true’ is uncommented; otherwise, it contains no value.
`-I$DB2PATH/include`	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
`-c`	Perform compile only; no link. Compile and link are separate steps.
Link Options:	
`xlc_r`	Use the multi-threaded version of the compiler as a front end for the linker.
`$EXTRA_CFLAG`	Contains ″-q64″ value if ’BUILD_64BIT=true’ is uncommented; otherwise, it contains no value.
`-qmkshrobj`	Create the shared library.
`-o $1`	Specify the executable program.
`$1.o`	Specify the object file.
`utilcli.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 runtime shared libraries. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, the compiler assumes the following path: `/usr/lib:/lib`.
`-ldb2`	Link with the DB2 library.
`-bE:$.exp`	Specify an export file. The export file contains a list of routines.
Refer to your compiler documentation for additional compiler options.	

Related tasks:

- "Building CLI Routines on UNIX" on page 255
- "Building CLI Stored Procedures with Configuration Files" on page 263

Related reference:

- "AIX CLI Application Compile and Link Options" on page 258

Related samples:

- "bldrtn -- Builds AIX CLI routines (stored procedures and UDFs)"

CLI Routines and Configuration Files on AIX

The following section describes how to build CLI routines with configuration files. This method of building CLI routines is only available on AIX using the VisualAge C++ compiler.

Building CLI Stored Procedures with Configuration Files: The configuration file, `clis.icc`, in `sqllib/samples/cli`, allows you to build DB2 CLI stored procedures.

Procedure:

To use the configuration file to build the DB2 CLI stored procedure `spserver` from the source file `spserver.c`, do the following:

1. Set the CLIS environment variable to the program name by entering:
   ```
   export CLIS=spserver
   ```
2. If you have a `clis.ics` file in your working directory, produced by building a different program with the `clis.icc` file, delete the `clis.ics` file with this command:
   ```
   rm clis.ics
   ```

   An existing `clis.ics` file produced for the same program you are going to build again does not have to be deleted.
3. Compile the sample program by entering:
   ```
   vacbld clis.icc
   ```

   Note: The `vacbld` command is provided by VisualAge C++.

The stored procedure is copied to the server in the path `sqllib/function`.

Next, catalog the stored procedures by running the `spcreate.db2` script on the server. First, connect to the database with the user ID and password of the instance where the database is located:

```
db2 connect to sample userid password
```

If the stored procedures were previously cataloged, you can drop them with this command:

```
db2 -td@ -vf spdrop.db2
```

Then catalog them with this command:

```
db2 -td@ -vf spcreate.db2
```

Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the stored procedure `spserver`, you can build the CLI client application `spclient` that calls the stored procedure. You can build `spclient` by using the configuration file, `cli.icc`.

To call the stored procedure, run the sample client application by entering:

`spclient` *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be `sample`, or its remote alias, or some other name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the shared library, `spserver`, and executes a number of stored procedure functions on the server database. The output is returned to the client application.

Related tasks:
- "Building CLI Routines on UNIX" on page 255

Related samples:
- "spclient.c -- Call various stored procedures (CLI)"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "

HP-UX

Build Script for HP-UX Applications

The following is the `bldapp` script for building CLI applications on HP-UX:

```
#! /bin/sh

# SCRIPT: bldapp
# Builds HP-UX CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc $EXTRA_CFLAG -Ae -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc $EXTRA_CFLAG -Ae -I$DB2PATH/include -c $1.c

# Link the program.
cc $EXTRA_CFLAG -o $1 $1.o utilcli.o -L$DB2PATH/lib -ldb2
```

HP-UX CLI Application Compile and Link Options

These compile and link options are recommended by DB2 for building CLI applications with the HP-UX C compiler. They are demonstrated in the `sqllib/samples/cli/bldapp` build script.

Compile and Link Options for bldapp

Compile Options:

cc Use the C compiler.

$EXTRA_CFLAG
Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64=bit, the value is `+DA2.0W`.

+DAportable (32-bit only)
Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information.

+DA2.0W (64-bit only)
Must be used to generate 64-bit code.

-Ae Enables HP ANSI extended mode.

-I$DB2PATH/include
Specifies the location of the DB2 include files.

-I$DB2PATH/include
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

-c Perform compile only; no link. Compile and link are separate steps.

Compile and Link Options for bldapp	
Link Options:	
`cc`	Use the compiler as a front end for the linker.
`$EXTRA_CFLAG`	Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64=bit, the value is `+DA2.0W`.
`+DAportable` (32-bit only)	Use code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information.
`+DA2.0W` (64-bit only)	Must be used to generate 64-bit code.
`-o $1`	Specify the executable program.
`-o $1.o`	Specify the object file.
`utilcli.o`	Include the utility object file for error checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 runtime shared libraries. For example, `$HOME/sqllib/lib`
`-ldb2`	Link with the database manager library.
Refer to your compiler documentation for additional compiler options.	

Note: The `+DAportable` option comes with a slight performance cost. To improve performance, you can remove the `+DAportable` option from the build files and makefiles provided in the `sqllib/samples` directory. Without this option, you may get a warning similar to the following when building HP-UX programs:

```
(Warning) At least one PA 2.0 object file (<filename>.o) was detected.
The linked object may not run on a PA 1.x system.
```

where `<filename>` is the program file you are compiling.

Unless you have a PA_RISC 1.1 or 2.0 system, this warning does not apply.

Related tasks:

- "Building CLI Applications on UNIX" on page 253

Related reference:

- "HP-UX CLI Routine Compile and Link Options" on page 268

Related samples:

- "bldapp -- Builds HP-UX C applications"

Build Script for HP-UX Routines

The following is the `bldrtn` script for building CLI routines on HP-UX:

```
#! /bin/sh

# SCRIPT: bldrtn
# Builds HP-UX CLI routines (stored procedures and UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true
if [ "$BUILD_64BIT" != "" ]
then
  EXTRA_CFLAG="+DA2.0W"
else
  EXTRA_CFLAG="+DAportable"
fi

# Compile the error-checking utility.
cc $EXTRA_CFLAG +u1 +z -Ae -I$DB2PATH/include \
  -D_POSIX_C_SOURCE=199506L -c utilcli.c

# Compile the program.
cc $EXTRA_CFLAG +u1 +z -Ae -I$DB2PATH/include \
  -D_POSIX_C_SOURCE=199506L -c $1.c

# Link the program.
ld -b -o $1 $1.o utilcli.o -L$DB2PATH/lib \
  -ldb2 -lpthread

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

HP-UX CLI Routine Compile and Link Options

These compile and link options are recommended by DB2 for building CLI routines with the HP-UX C compiler. They are demonstrated in the `sqllib/samples/cli/bldrtn` build script.

Compile and Link Options for bldrtn

Compile Options:

`cc` The C compiler.

`$EXTRA_CFLAG`
: Contains different values depending on whether 32-bit or 64-bit support is enabled. For 32-bit, the value is `+DAportable`; for 64=bit, the value is `+DA2.0W`.

`+DAportable` (32-bit only)
: Generates code compatible across PA_RISC 1.x and 2.0 workstations and servers. This option should be used if the application's portability is a concern. Building without this option will generate code better optimized for the processor level of the building machine, but won't work on older PA-RISC processor levels. See the compiler documentation for more information.

`+DA2.0W` (64-bit only)
: Must be used to generate 64-bit code.

`+u1` Allow unaligned data access. Use only if your application uses unaligned data.

`+z` Generate position-independent code.

`-Ae` Enables HP ANSI extended mode.

`-I$DB2PATH/include`
: Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

`-D_POSIX_C_SOURCE=199506L`
: POSIX thread library option that ensures _REENTRANT is defined, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).

`-c` Perform compile only; no link. Compile and link are separate steps.

Link Options:

`ld` Use the linker to link.

`-b` Create a shared library rather than a normal executable.

`-o $1` Specify the executable.

`$1.o` Specify the object file.

`utilcli.o`
: Link in the error-checking utility object file.

`-L$DB2PATH/lib`
: Specify the location of the DB2 runtime shared libraries. For example: `-L$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.

`-ldb2` Link with the DB2 library.

`-lpthread`
: Link with the POSIX thread library.

Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building CLI Routines on UNIX" on page 255

Related reference:

- "HP-UX CLI Application Compile and Link Options" on page 265

Related samples:

- "bldrtn -- Builds HP-UX C routines (stored procedures and UDFs)"

Linux

Build Script for Linux Applications

The following is the `bldapp` script for building CLI applications on Linux:

```
#! /bin/sh

# SCRIPT: bldapp
# Builds Linux CLI applications
# Usage: bldapp <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-Wl,-rpath$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
gcc -I$DB2PATH/include -c utilcli.c

# Compile the program.
gcc -I$DB2PATH/include -c $1.c

# Link the program.
gcc -o $1 $1.o utilcli.o $EXTRA_LFLAG -L$DB2PATH/lib -ldb2
```

Linux CLI Application Compile and Link Options

These are the compile and link options recommended by DB2 for building CLI applications with the GNU/Linux gcc compiler. They are demonstrated in the `sqllib/samples/cli/bldapp` build script.

Compile and Link Options for bldapp
Compile Options:
`gcc` The C compiler.
`-I$DB2PATH/include` Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`
`-c` Perform compile only; no link. Compile and link are separate steps.
Link options:
`gcc` Use the compiler as a front end for the linker.
`-o $1` Specify the executable.
`$1.o` Include the program object file.
`utilcli.o` Include the utility object file for error checking.
`-L$DB2PATH/lib` Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`$EXTRA_LFLAG` If 'RUNTIME=true' is uncommented, contains value "-Wl,-rpath,$DB2PATH/lib"; otherwise, contains no value.
`-Wl,-rpath,$DB2PATH/lib` Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2` Link with the DB2 library.
Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building CLI Applications on UNIX" on page 253

Related reference:

- "Linux CLI Routine Compile and Link Options" on page 272

Related samples:

- "bldapp -- Builds Linux C applications"

Build Script for Linux Routines

The following is the `bldrtn` script for building CLI routines on Linux:

```
#! /bin/sh

# SCRIPT: bldrtn
# Builds Linux CLI routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# Set the runtime path since routines run as setuid.
EXTRA_LFLAG="-Wl,-rpath,$DB2PATH/lib"

# Compile the error-checking utility.
gcc -fpic -I$DB2PATH/include -c utilcli.c -D_REENTRANT

# Compile the program.
gcc -fpic -I$DB2PATH/include -c $1.c -D_REENTRANT

# Link the program.
gcc -o $1 $1.o utilcli.o -shared $EXTRA_LFLAG \
  -L$DB2PATH/lib -ldb2 -lpthread

# Copy the shared library to the function subdirectory.
# The user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Linux CLI Routine Compile and Link Options

These are the compile and link options recommended by DB2 for building CLI routines with the GNU/Linux gcc compiler. They are demonstrated in the `sqllib/samples/cli/bldrtn` build script.

Compile and Link Options for bldrtn	
Compile Options:	
`gcc`	The C compiler.
`-fpic`	Allows position independent code.
`-I$DB2PATH/include`	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
`-c`	Perform compile only; no link. Compile and link are separate steps.
`-D_REENTRANT`	Defines _REENTRANT, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).

Compile and Link Options for bldrtn
Link Options:
`gcc` Use the compiler as a front end for the linker.
`-o $1` Specify the executable.
`$1.o` Include the program object file.
`utilcli.o` Include the utility object file for error-checking.
`-shared` Generate a shared library.
`-L$DB2PATH/lib` Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`$EXTRA_LFLAG` Contains the value ″-Wl,-rpath,$DB2PATH/lib″ to set the runtime path since routines run as setuid.
`-Wl,-rpath,$DB2PATH/lib` Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2` Link with the DB2 library.
`-lpthread` Link with the POSIX thread library.
Refer to your compiler documentation for additional compiler options.

Related tasks:

- "Building CLI Routines on UNIX" on page 255

Related reference:

- "Linux CLI Application Compile and Link Options" on page 270

Related samples:

- "bldrtn -- Builds Linux C routines (stored procedures or UDFs)"

Solaris

Build Script for Solaris Applications

The following is the `bldapp` script for building CLI applications on Solaris:

```
#! /bin/sh

# SCRIPT: bldapp
# Builds Solaris CLI applications
# Usage: bldapp <prog_name> [ <db_name> [ <userid> <password> ]]

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# To specify a runtime path for shared libraries, uncomment the
# following line (usually only needed for setuid applications).
# RUNTIME=true

if [ "$RUNTIME" != "" ]
then
  EXTRA_LFLAG="-R$DB2PATH/lib"
else
  EXTRA_LFLAG=""
fi

# If an embedded SQL program, precompile and bind it.
if [ -f $1".sqc" ]
then
  ./embprep $1 $2 $3 $4
fi

# Compile the error-checking utility.
cc -xarch=$CFLAG_ARCH -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -xarch=$CFLAG_ARCH -I$DB2PATH/include  -c $1.c

# Link the program.
cc -xarch=$CFLAG_ARCH -mt -o $1 $1.o utilcli.o \
  -L$DB2PATH/lib $EXTRA_LFLAG -ldb2
```

Solaris CLI Application Compile and Link Options

These are the compile and link options recommended by DB2 for building CLI applications with the Forte C compiler. They are demonstrated in the `sqllib/samples/cli/bldapp` build script.

Compile and Link Options for bldapp

Compile Options:

`cc` Use the C compiler.

`-xarch=$CFLAG_ARCH`
This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. See the note below for more information.

`-I$DB2PATH/include`
Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`

`-c` Perform compile only; no link. This script has separate compile and link steps.

Link Options:

`cc` Use the compiler as a front end for the linker.

`-xarch=$CFLAG_ARCH`
This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. See the note below for more information.

`-mt` Link in multi-thread support to prevent problems calling fopen.
Note: If POSIX threads are used, DB2 applications also have to link with `-lpthread`, whether or not they are threaded.

`-o $1` Specify the executable program.

`$1.o` Include the program object file.

`utilcli.o`
Include the utility object file for error checking.

`-L$DB2PATH/lib`
Specify the location of the DB2 static and shared libraries at link-time. For example, `$HOME/sqllib/lib`

`$EXTRA_LFLAG`
If ′RUNTIME=true′ is uncommented, contains value ″-R$DB2PATH/lib″; otherwise, contains no value.

`-R$DB2PATH/lib`
Specify the location of the DB2 shared libraries at run-time. For example, `$HOME/sqllib/lib`

`-ldb2` Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

Note: The -xarch option provides a workaround for problems with executables produced by the Forte C compiler, like the following:

1. syntax error at line 1: `(' unexpected
2. ksh: <application name>: cannot execute (where application name is the name of the compiled executable)

Related tasks:

- "Building CLI Applications on UNIX" on page 253

Related reference:

- "Solaris CLI Routine Compile and Link Options" on page 277

Related samples:

- "bldapp -- Builds Solaris C applications"

Build Script for Solaris Routines

The following is the `bldrtn` script for building CLI routines on Solaris:

```
#! /bin/sh

# SCRIPT: bldrtn
# Builds Solaris CLI routines (stored procedures or UDFs)
# Usage: bldrtn <prog_name>

# Set DB2PATH to where DB2 will be accessed.
# The default is the standard instance path.
DB2PATH=$HOME/sqllib

# To compile 64 bit programs, uncomment the following line.
# BUILD_64BIT=true

if [ "$BUILD_64BIT" != "" ]
then
  CFLAG_ARCH=v9
else
  CFLAG_ARCH=v8plusa
fi

# Set the runtime path since routines run as setuid.
EXTRA_LFLAG="-R$DB2PATH/lib"

# Compile the error-checking utility.
cc -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -Kpic \
  -I$DB2PATH/include -c utilcli.c

# Compile the program.
cc -xarch=$CFLAG_ARCH -mt -DUSE_UI_THREADS -Kpic \
  -I$DB2PATH/include -c $1.c

# Link the program.
cc -xarch=$CFLAG_ARCH -mt -G -o $1 $1.o utilcli.o \
  -L$DB2PATH/lib $EXTRA_LFLAG -ldb2

# Copy the shared library to the sqllib/function subdirectory.
# Note: the user must have write permission to this directory.
rm -f $DB2PATH/function/$1
cp $1 $DB2PATH/function
```

Solaris CLI Routine Compile and Link Options

These are the compile and link options recommended by DB2 for building CLI routines with the Forte C compiler. They are demonstrated in the `sqllib/samples/cli/bldrtn` build script.

Compile and Link Options for bldrtn	
Compile options:	
`cc`	The C compiler.
`-xarch=$CFLAG_ARCH`	This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. See the note below for more information.
`-mt`	Allow multi-threaded support, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
`-DUSE_UI_THREADS`	Allows Sun's ″UNIX International″ threads APIs.
`-Kpic`	Generate position-independent code for shared libraries.
`-I$DB2PATH/include`	Specify the location of the DB2 include files. For example: `$HOME/sqllib/include`.
`-c`	Perform compile only; no link. Compile and link are separate steps.

Compile and Link Options for bldrtn

Link Options:

`cc`	Use the compiler as a front end for the linker.
`-xarch=$CFLAG_ARCH`	This option ensures that the compiler will produce valid executables when linking with `libdb2.so`. The value for $CFLAG_ARCH is set to either ″v8plusa″ for 32-bit, or ″v9″ for 64-bit. See the note below for more information.
`-mt`	Allow multi-threaded support, needed as the routines may run in the same process as other routines (THREADSAFE) or in the engine itself (NOT FENCED).
`-G`	Generate a shared library.
`-o $1`	Specify the executable.
`$1.o`	Include the program object file.
`utilcli.o`	Include the utility object file for error-checking.
`-L$DB2PATH/lib`	Specify the location of the DB2 static and shared libraries at link-time. For example: `$HOME/sqllib/lib`. If you do not specify the `-L` option, `/usr/lib:/lib` is assumed.
`$EXTRA_LFLAG`	Contains the value ″-R$DB2PATH/lib″ to set the runtime path since routines run as setuid.
`-R$DB2PATH/lib`	Specify the location of the DB2 shared libraries at run-time. For example: `$HOME/sqllib/lib`.
`-ldb2`	Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

Note: The -xarch option provides a workaround for problems with executables produced by the Forte C compiler, like the following:

1. syntax error at line 1: \`(′ unexpected
2. ksh: <application name>: cannot execute (where application name is the name of the compiled executable)

Related tasks:

- “Building CLI Routines on UNIX” on page 255

Related reference:

- “Solaris CLI Application Compile and Link Options” on page 274

Related samples:

- “bldrtn -- Builds Solaris C routines (stored procedures or UDFs)”

Windows

The following sections describe how to build CLI applications and routines on the Windows platform, and provide sample build scripts for each platform with descriptions of their compile and link options.

Building CLI Applications on Windows

DB2 provides batch files for compiling and linking CLI programs. These are located in the `sqllib\samples\cli` directory, along with sample programs that can be built with these files.

The batch file `bldapp.bat` contains the commands to build a DB2 CLI program. It takes up to four parameters, represented inside the batch file by the variables `%1`, `%2`, `%3`, and `%4`.

The parameter, `%1`, specifies the name of your source file. This is the only required parameter, and the only one needed for CLI programs that do not contain embedded SQL. Building embedded SQL programs requires a connection to the database so three optional parameters are also provided: the second parameter, `%2`, specifies the name of the database to which you want to connect; the third parameter, `%3`, specifies the user ID for the database, and `%4` specifies the password.

If the program contains embedded SQL, indicated by the `.sqc` or `.sqx` extension, then the `embprep.bat` batch file is called to precompile the program, producing a program file with either a `.c` or a `.cxx` extension, respectively.

Procedure:

The following examples show you how to build and run CLI applications.

To build the sample program `tbinfo` from the source file `tbinfo.c`, enter:

```
bldapp tbinfo
```

The result is an executable file `tbinfo`. You can run the executable file by entering the executable name:

```
tbinfo
```

Building and Running Embedded SQL Applications

There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sqc`:

1. If connecting to the sample database on the same instance, enter:
   ```
   bldapp dbusemx
   ```
2. If connecting to another database on the same instance, also enter the database name:
   ```
   bldapp dbusemx database
   ```
3. If connecting to a database on another instance, also enter the user ID and password of the database instance:
   ```
   bldapp dbusemx database userid password
   ```

The result is an executable file, `dbusemx`.

There are three ways to run this embedded SQL application:

1. If accessing the `sample` database on the same instance, simply enter the executable name:
   ```
   dbusemx
   ```
2. If accessing another database on the same instance, enter the executable name and the database name:
   ```
   dbusemx database
   ```
3. If accessing a database on another instance, enter the executable name, database name, and user ID and password of the database instance:
   ```
   dbusemx database userid password
   ```

Related tasks:

- "Setting Up the CLI Environment" on page 247
- "Setting Up the Windows CLI Environment" on page 250
- "Building CLI Routines on Windows" on page 281

Related reference:

- "Windows CLI Application Compile and Link Options" on page 283

Related samples:

- "bldapp.bat -- Builds C applications on Windows"
- "embprep.bat -- Prep and binds a C/C++ or Micro Focus COBOL embedded SQL program on Windows"
- "dbusemx.sqc -- How to execute embedded SQL statements in CLI (CLI)"
- "tbinfo.c -- How to get information about tables from the system catalog tables (CLI)"

Building CLI Routines on Windows

DB2 provides batch files for compiling and linking CLI programs. These are located in the `sqllib\samples\cli` directory, along with sample programs that can be built with these files.

The batch file `bldrtn.bat` contains the commands to build CLI routines (stored procedures and user-defined functions). `bldrtn.bat` creates a DLL on the server. It takes one parameter, represented inside the batch file by the variable `%1`, which specifies the name of your source file. The batch file uses the source file name for the DLL name.

Procedure:

To build the `spserver` DLL from the source file `spserver.c`:

1. Enter the batch file name and program name:

   ```
   bldrtn spserver
   ```

   The batch file uses the module definition file `spserver.def`, contained in the same directory as the CLI sample programs, to build the DLL. The batch file then copies the DLL, `spserver.dll`, to the server in the path `sqllib\function`.
2. Next, catalog the routines by running the `spcat` script on the server:

   ```
   spcat
   ```

   This script connects to the sample database, uncatalogs the routines if they were previously cataloged by calling `spdrop.db2`, then catalogs them by calling `spcreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spcreate.db2` scripts individually.
3. Then, stop and restart the database to allow the new shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Once you build the DLL `spserver`, you can build the CLI client application `spclient` that calls the routines within it.

You can build `spclient` by using the script file, `blapp`.

To call the routines, run the sample client application by entering:

`spclient` *database userid password*

where

database
Is the name of the database to which you want to connect. The name could be `sample`, or its alias, or another database name.

userid Is a valid user ID.

password
Is a valid password.

The client application accesses the DLL, `spserver`, which executes the routines on the server database. The output is returned to the client application.

Related tasks:

- "Setting Up the CLI Environment" on page 247
- "Setting Up the Windows CLI Environment" on page 250
- "Building CLI Applications on Windows" on page 279

Related reference:

- "Windows CLI Routine Compile and Link Options" on page 285

Related samples:

- "bldapp.bat -- Builds C applications on Windows"
- "bldrtn.bat -- Builds C routines (stored procedures and UDFs) on Windows"
- "spcat -- To catalog C stored procedures on UNIX"
- "spcreate.db2 -- How to catalog the stored procedures contained in spserver.sqc "
- "spdrop.db2 -- How to uncatalog the stored procedures contained in spserver.sqc "
- "spclient.c -- Call various stored procedures (CLI)"
- "spserver.c -- Definition of various types of stored procedures (CLI)"

Build Script for Windows Applications

The following is the `bldapp` script for building CLI applications on Windows:

```
@echo off

rem BATCH FILE: bldapp.bat
rem Builds Windows CLI applications
rem Usage: bldapp prog_name [ db_name [ userid password ]]

rem Default compiler is set to Microsoft Visual C++
rem To use a different compiler, comment out 'set BLDCOMP=cl'
rem and uncomment 'set BLDCOMP=icl'áor 'set BLDCOMP=ecl'
rem Microsoft C/C++ Compiler
set BLDCOMP=cl

rem Intel C++ Compiler for 32-bit applications
rem set BLDCOMP=icl

rem Intel C++ Compiler for Itanium 64-bit applications
rem set BLDCOMP=ecl

if exist "%1.sqc" call embprep %1 %2 %3 %4
if exist "%1.sqx" call embprep %1 %2 %3 %4

rem Compile the error-checking utility.
%BLDCOMP% -Zi -Od -c -W1 -DWIN32 utilcli.c

rem  Compile the program.
if exist "%1.sqx" goto cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.c
goto link_step
:cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.cxx

rem  Link the program.
:link_step
link -debug -out:%1.exe %1.obj utilcli.obj db2cli.lib db2api.lib
@echo on
```

Windows CLI Application Compile and Link Options

These compile and link options are recommended by DB2 for building CLI applications with the Microsoft Visual C++ compiler. They are demonstrated in the `sqllib\samples\cli\bldapp.bat` batch file.

Compile and Link Options for bldapp	
Compile Options:	
%BLDCOMP%	Variable for the compiler. The default is `cl`, the Microsoft Visual C++ compiler. It can be also set to `icl`, the Intel C++ Compiler for 32-bit applications, or `ecl`, the Intel C++ Compiler for Itanium 64-bit applications.
-Zi	Enable debugging information.
-Od	Disable optimizations. It is easier to use a debugger with optimization off.
-c	Perform compile only; no link. This book assumes that compile and link are separate steps.
-W2	Set warning level.
-DWIN32	Compiler option necessary for Windows operating systems.

Compile and Link Options for bldapp	
Link Options:	
link	Use the 32-bit linker.
-debug	Include debugging information.
-out:%1.exe	Specify the executable.
%1.obj	Include the object file.
utilcli.obj	Include the utility object file for error checking.
db2cli.lib	Link with the DB2 CLI library.
Refer to your compiler documentation for additional compiler options.	

Related tasks:
- "Building CLI Applications on Windows" on page 279
- "Building CLI Routines on Windows" on page 281

Related samples:
- "bldapp.bat -- Builds C applications on Windows"

Build Script for Windows Routines

The following is the `bldrtn` script for building CLI routines on Windows:

```
@echo off

rem BATCH FILE: bldrtn.bat
rem Builds Windows CLI routines (stored procedures and UDFs)
rem using the Microsoft Visual C++ compiler
rem Usage: bldrtn prog_name

rem Default compiler is set to Microsoft Visual C++
rem To use a different compiler, comment out 'set BLDCOMP=cl'
rem and uncomment 'set BLDCOMP=icl'áor 'set BLDCOMP=ecl'
rem Microsoft C/C++ Compiler
set BLDCOMP=cl

rem Intel C++ Compiler for 32-bit applications
rem set BLDCOMP=icl

rem Intel C++ Compiler for Itanium 64-bit applications
rem set BLDCOMP=ecl

if "%1" == "" goto error

rem Compile the program.
if exist "%1.cxx" goto cpp
```

```
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.c  utilcli.c
goto link_step
:cpp
%BLDCOMP% -Zi -Od -c -W2 -DWIN32 %1.cxx  utilcli.c

rem Link the program.
:link_step
link -debug -dll -out:%1.dll %1.obj utilcli.obj db2cli.lib -def:%1.def

rem Copy the stored procedure DLL to the 'function' directory
copy %1.dll "%DB2PATH%\function"

goto exit
:error
echo Usage: bldrtn prog_name
:exit
@echo on
```

Windows CLI Routine Compile and Link Options

These compile and link options are recommended by DB2 for building CLI routines with the Microsoft Visual C++ compiler. They are demonstrated in the `sqllib\samples\cli\bldrtn.bat` batch file.

Compile and Link Options for bldrtn	
Compile Options:	
%BLDCOMP%	Variable for the compiler. The default is `cl`, the Microsoft Visual C++ compiler. It can be also set to `icl`, the Intel C++ Compiler for 32-bit applications, or `ecl`, the Intel C++ Compiler for Itanium 64-bit applications.
-Zi	Enable debugging information
-Od	Disable optimizations. It is easier to use a debugger with optimization off.
-c	Perform compile only; no link. The batch file has separate compile and link steps.
-W2	Set warning level.
-DWIN32	Compiler option necessary for Windows operating systems.

Compile and Link Options for bldrtn
Link Options:

`link`	Use the 32-bit linker.
`-debug`	Include debugging information.
`-OUT:%1.dll`	Build a .DLL file.
`%1.obj`	Include the object file.
`utilcli.obj`	Include the utility object file for error-checking.
`db2cli.lib`	Link with the DB2 CLI library.
`-def:%1.def`	Use the module definition file.

Refer to your compiler documentation for additional compiler options.

Related tasks:
- "Building CLI Applications on Windows" on page 279
- "Building CLI Routines on Windows" on page 281

Related samples:
- "bldrtn.bat -- Builds C routines (stored procedures and UDFs) on Windows"

Chapter 24. CLI Sample Programs

CLI Sample Programs

DB2 CLI includes various sample applications located in the following location:

- On Windows® operating systems: `%DB2PATH%\sqllib\samples\cli` (where `%DB2PATH%` is a variable that determines where DB2® is installed)
- On UNIX: `$HOME/sqllib/samples/cli` (where $HOME is the home directory of the instance owner)

The `README` file in the same directory lists each sample along with an explanation, and describes how to build the samples using the makefile and build files provided.

Related tasks:

- "Setting Up the CLI Environment" on page 247
- "Building CLI Applications on UNIX" on page 253
- "Building CLI Routines on UNIX" on page 255
- "Building CLI Applications on Windows" on page 279
- "Building CLI Routines on Windows" on page 281

Related reference:

- "DB2 CLI Samples" on page 287

DB2 CLI Samples

Table 19. Sample CLI Program Files

Sample Program Name	Program Description
Tutorial Samples - Programs that demonstrate basic database operations.	
`tut_mod.c`	How to modify table data.
`tut_read.c`	How to read tables.
`tut_use.c`	How to use a database.
Installation Image Level - Samples that deal with the installation image level of DB2 and CLI.	
`ilinfo.c`	How to get and set installation level information (such as the version of the CLI driver).
Client Level - Samples that deal with the client level of DB2.	

Table 19. Sample CLI Program Files (continued)

Sample Program Name	**Program Description**
`cli_info.c`	How to get and set client level information.
`clihandl.c`	How to allocate and free handles.
`clisqlca.c`	How to work with SQLCA data.
Instance Level - Samples that deal with the instance level of DB2.	
`ininfo.c`	How to get and set instance level information.
Database Level - Samples that deal with database objects in DB2.	
`dbcongui.c`	How to connect to a database with a Graphical User Interface (GUI).
`dbconn.c`	How to connect and disconnect from a database.
`dbinfo.c`	How to get and set information at a database level.
`dbmcon.c`	How to connect and disconnect from multiple databases.
`dbmconx.c`	How to connect and disconnect from multiple databases with embedded SQL.
`dbmconx1.h`	Header file for dbmconx1.sqc.
`dbmconx1.sqc`	Embedded SQL file for the dbmconx program.
`dbmconx2.h`	Header file for dbmconx2.sqc.
`dbmconx2.sqc`	Embedded SQL file for the dbmconx program.
`dbnative.c`	How to translate a statement that contains an ODBC escape clause to a data source specific format.
`dbuse.c`	How to use database objects.
`dbusemx.sqc`	How to use database objects with embedded SQL.
Table Level - Samples that deal with table objects in DB2.	
`tbconstr.c`	How to work with table constraints.
`tbcreate.c`	How to create, alter, and drop tables.
`tbinfo.c`	How to get and set information at a table level.
`tbmod.c`	How to modify information in a table.
`tbread.c`	How to read information in a table.
Data Type Level - Samples that deal with data types.	
`dtinfo.c`	How to get information about data types.
`dtlob.c`	How to read and write LOB data.
`dtudt.c`	How to create, use, and drop user defined distinct types.
Stored Procedure Level - Samples that demonstrate stored procedures.	
`spcat`	Stored procedure catalog script for the spserver program. This script calls spdrop.db2 and spcreate.db2.

Table 19. Sample CLI Program Files (continued)

Sample Program Name	Program Description
`spcreate.db2`	CLP script to issue CREATE PROCEDURE statements.
`spdrop.db2`	CLP script to drop stored procedures from the catalog.
`spclient.c`	Client program used to call the server functions declared in spserver.c.
`spserver.c`	Stored procedure functions built and run on the server.
`spclires.c`	Client application that demonstrates the difference between SQLMoreResults and SQLNextResults for multiple result sets.
`spcall.c`	Client program for calling any stored procedure.
UDF Level - Samples that demonstrate user defined functions.	
`udfcli.c`	Client application which calls the user defined function in udfsrv.c.
`udfsrv.c`	User defined function ScalarUDF called by udfcli.c.
Common Utility Files	
`utilcli.c`	Utility functions used in CLI samples.
`utilcli.h`	Header file for utility functions used in CLI samples.

Related concepts:

- "Sample Files" in the *Application Development Guide: Building and Running Applications*
- "Sample Programs: Structure and Design" in the *Application Development Guide: Building and Running Applications*

Part 4. CLI/ODBC Configuration Keywords

Chapter 25. CLI/ODBC Configuration Keywords

CLI/ODBC configuration keywords allow you to customize the behavior of the DB2 CLI driver. This chapter describes setting these keywords through the db2cli.ini initialization file and contains a listing of available configuration keywords.

db2cli.ini Initialization File

The db2cli.ini initialization file contains various keywords and values that can be used to configure the behavior of DB2 CLI and the applications using it. The keywords are associated with the database *alias name*, and affect all DB2 CLI and ODBC applications that access the database.

By default, the location of the DB2 CLI/ODBC configuration keyword file is in the sqllib directory on Window platforms, and in the sqllib/cfg directory of the database instance running the CLI/ODBC applications on UNIX® platforms. If the ODBC Driver Manager is used to configure a User Data Source on the Windows® platform, a db2cli.ini may be created in the user's home (profile) directory.

The environment variable *DB2CLIINIPATH* can also be used to override the default and specify a different location for the file.

The configuration keywords enable you to:
- Configure general features such as data source name, user name, and password.
- Set options that will affect performance.
- Indicate query parameters such as wild card characters.
- Set patches or work-arounds for various ODBC applications.
- Set other, more specific features associated with the connection, such as code pages and IBM® Graphic data types.

The db2cli.ini initialization file is an ASCII file which stores values for the DB2 CLI configuration options. A sample file is shipped to help you get started.

There is one section within the file for each database (data source) the user wishes to configure. If needed, there is also a common section that affects all connections to DB2.

Only the keywords that apply to all connections to DB2 through the DB2 CLI/ODBC driver are included in the COMMON section. This includes the following keywords:

- DISABLEMULTITHREAD
- JDBCTRACE
- JDBCTRACEFLUSH
- JDBCTRACEPATHNAME
- TRACE
- TRACECOMM
- TRACEFILENAME
- TRACEFLUSH
- TRACELOCKS
- TRACEPATHNAME
- TRACEPIDLIST
- TRACEPIDTID
- TRACEREFRESHINTERVAL
- TRACESTMTONLY
- TRACETIME
- TRACETIMESTAMP

All other keywords are to be placed in the database specific section, described below.

Note: Configuration keywords are valid in the COMMON section, however, they will apply to all database connections.

The COMMON section of the db2cli.ini file begins with:

```
[COMMON]
```

Before setting a common keyword it is important to evaluate its impact on all DB2 CLI/ODBC connections from that client. A keyword such as TRACE, for instance, will generate information on all DB2 CLI/ODBC applications connecting to DB2 on that client, even if you are intending to troubleshoot only one of those applications.

Each database specific section always begins with the name of the data source name (DSN) between square brackets:

```
[data source name]
```

This is called the ***section header***.

The parameters are set by specifying a keyword with its associated keyword value in the form:

```
KeywordName =keywordValue
```

- All the keywords and their associated values for each database must be located below the database section header.
- If the database-specific section does not contain a DBALIAS keyword, the data source name is used as the database alias when the connection is established. The keyword settings in each section apply only to the applicable database alias.
- The keywords are not case sensitive; however, their values can be if the values are character based.
- If a database is not found in the .INI file, the default values for these keywords are in effect.
- Comment lines are introduced by having a semicolon in the first position of a new line.
- Blank lines are permitted.
- If duplicate entries for a keyword exist, the first entry is used (and no warning is given).

The following is a sample .INI file with 2 database alias sections:

```
; This is a comment line.
[MYDB22]
AUTOCOMMIT=0
TABLETYPE="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CURRENTSQLID=SAAID
TABLETYPE="'TABLE'"
SCHEMALIST="'USER1',CURRENT SQLID,'USER2'"
```

Although you can edit the db2cli.ini file manually on all platforms, we recommend that you use the Configuration Assistant if it is available on your platform or the UPDATE CLI CONFIGURATION command. You must add a blank line after the last entry if you manually edit the db2cli.ini file.

Related reference:

- "UPDATE CLI CONFIGURATION" in the *Command Reference*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "DBALIAS CLI/ODBC Configuration Keyword" on page 313
- "TRACE CLI/ODBC Configuration Keyword" on page 339

CLI/ODBC Configuration Keywords Listing by Category

The CLI/ODBC configuration keywords can be divided into the following categories:

- "Compatibility Configuration Keywords"
- "Data Source Configuration Keywords"
- "Data Type Configuration Keywords"
- "Enterprise Configuration Keywords" on page 297
- "Environment Configuration Keywords" on page 297
- "File DSN Configuration Keywords" on page 297
- "Optimization Configuration Keywords" on page 298
- "Service Configuration Keywords" on page 298
- "Static SQL Configuration Keywords" on page 299
- "Transaction Configuration Keywords" on page 299

Compatibility Configuration Keywords:

The **Compatibility** set of options is used to define DB2 behavior. They can be set to ensure that other applications are compatible with DB2.

- "CONNECTCODEPAGE CLI/ODBC Configuration Keyword" on page 304
- "DEFERREDPREPARE CLI/ODBC Configuration Keyword" on page 316
- "DISABLEKEYSETCURSOR CLI/ODBC Configuration Keyword" on page 317
- "DISABLEMULTITHREAD CLI/ODBC Configuration Keyword" on page 317
- "DISABLEUNICODE CLI/ODBC Configuration Keyword" on page 318
- "UNICODESERVER CLI/ODBC Configuration Keyword" on page 352

Data Source Configuration Keywords:

General keywords.

- "DBALIAS CLI/ODBC Configuration Keyword" on page 313
- "PWD CLI/ODBC Configuration Keyword" on page 328
- "UID CLI/ODBC Configuration Keyword" on page 351

Data Type Configuration Keywords:

The **Data Type** set of options is used to define how DB2 reports and handles various data types.

- "BITDATA CLI/ODBC Configuration Keyword" on page 301

- “LOBMAXCOLUMNSIZE CLI/ODBC Configuration Keyword” on page 323
- “LONGDATACOMPAT CLI/ODBC Configuration Keyword” on page 323

Enterprise Configuration Keywords:

The **Enterprise** set of options is used to maximize the efficiency of connections to large databases.
- “CLISCHEMA CLI/ODBC Configuration Keyword” on page 303
- “CONNECTNODE CLI/ODBC Configuration Keyword” on page 305
- “CURRENTPACKAGESET CLI/ODBC Configuration Keyword” on page 307
- “CURRENTSCHEMA CLI/ODBC Configuration Keyword” on page 308
- “CURRENTSQLID CLI/ODBC Configuration Keyword” on page 309
- “DBNAME CLI/ODBC Configuration Keyword” on page 314
- “GRANTEELIST CLI/ODBC Configuration Keyword” on page 319
- “GRANTORLIST CLI/ODBC Configuration Keyword” on page 319
- “SCHEMALIST CLI/ODBC Configuration Keyword” on page 330
- “SYSSCHEMA CLI/ODBC Configuration Keyword” on page 337
- “TABLETYPE CLI/ODBC Configuration Keyword” on page 337

Environment Configuration Keywords:

The **Environment** set of options is used to define environment-specific settings, such as the location of various files on the server and client machines.
- “CURRENTFUNCTIONPATH CLI/ODBC Configuration Keyword” on page 306
- “DEFAULTPROCLIBRARY CLI/ODBC Configuration Keyword” on page 315
- “QUERYTIMEOUTINTERVAL CLI/ODBC Configuration Keyword” on page 329
- “TEMPDIR CLI/ODBC Configuration Keyword” on page 338

File DSN Configuration Keywords:

The **File DSN** set of options is used to set the TCP/IP settings for a file DSN connection.
- “DATABASE CLI/ODBC Configuration Keyword” on page 310
- “HOSTNAME CLI/ODBC Configuration Keyword” on page 320
- “PORT CLI/ODBC Configuration Keyword” on page 327
- “PROTOCOL CLI/ODBC Configuration Keyword” on page 328

- "SERVICENAME CLI/ODBC Configuration Keyword" on page 331

Optimization Configuration Keywords:

The **Optimization** set of options is used to speed up and reduce the amount of network flow between the CLI/ODBC Driver and the server.

- "DB2DEGREE CLI/ODBC Configuration Keyword" on page 311
- "DB2EXPLAIN CLI/ODBC Configuration Keyword" on page 312
- "DB2OPTIMIZATION CLI/ODBC Configuration Keyword" on page 313
- "KEEPSTATEMENT CLI/ODBC Configuration Keyword" on page 322
- "OPTIMIZEFORNROWS CLI/ODBC Configuration Keyword" on page 325
- "SKIPTRACE CLI/ODBC Configuration Keyword" on page 332
- "UNDERSCORE CLI/ODBC Configuration Keyword" on page 351

Service Configuration Keywords:

The **Service** set of options is used to help in troubleshooting problems with CLI/ODBC connections. Some options can also be used by programmers to gain a better understanding of how their CLI programs are translated into calls to the server.

- "APPENDAPINAME CLI/ODBC Configuration Keyword" on page 299
- "IGNOREWARNINGS CLI/ODBC Configuration Keyword" on page 321
- "IGNOREWARNLIST CLI/ODBC Configuration Keyword" on page 321
- "PATCH1 CLI/ODBC Configuration Keyword" on page 325
- "PATCH2 CLI/ODBC Configuration Keyword" on page 326
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACECOMM CLI/ODBC Configuration Keyword" on page 340
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACELOCKS CLI/ODBC Configuration Keyword" on page 343
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343
- "TRACEPIDLIST CLI/ODBC Configuration Keyword" on page 344
- "TRACEPIDTID CLI/ODBC Configuration Keyword" on page 345
- "TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword" on page 346
- "TRACESTMTONLY CLI/ODBC Configuration Keyword" on page 347
- "TRACETIME CLI/ODBC Configuration Keyword" on page 348
- "TRACETIMESTAMP CLI/ODBC Configuration Keyword" on page 349
- "WARNINGLIST CLI/ODBC Configuration Keyword" on page 353

Static SQL Configuration Keywords:

The **Static SQL** set of options is used when running static SQL statements in CLI/ODBC applications.
- "STATICCAPFILE CLI/ODBC Configuration Keyword" on page 333
- "STATICLOGFILE CLI/ODBC Configuration Keyword" on page 334
- "STATICMODE CLI/ODBC Configuration Keyword" on page 334
- "STATICPACKAGE CLI/ODBC Configuration Keyword" on page 335

Transaction Configuration Keywords:

The **Transaction** set of options is used to control and speed up SQL statements used in the application.
- "ASYNCENABLE CLI/ODBC Configuration Keyword" on page 300
- "AUTOCOMMIT CLI/ODBC Configuration Keyword" on page 300
- "CONNECTYPE CLI/ODBC Configuration Keyword" on page 306
- "CURSORHOLD CLI/ODBC Configuration Keyword" on page 309
- "KEEPCONNECT CLI/ODBC Configuration Keyword" on page 322
- "MODE CLI/ODBC Configuration Keyword" on page 324
- "SYNCPOINT CLI/ODBC Configuration Keyword" on page 336
- "TXNISOLATION CLI/ODBC Configuration Keyword" on page 350
- "USEOLDSTPCALL CLI/ODBC Configuration Keyword" on page 352

Related concepts:
- "db2cli.ini Initialization File" on page 293

APPENDAPINAME CLI/ODBC Configuration Keyword

Keyword Description:
Append the CLI/ODBC function name which generated an error to the error message text.

db2cli.ini Keyword Syntax:
APPENDAPINAME = **0** | **1**

Default Setting:
Do NOT display DB2 CLI function name.

Usage Notes:

The DB2 CLI function (API) name that generated an error is appended to the error message retrieved using `SQLGetDiagRec()` or `SQLError()`. The function name is enclosed in curly braces { }.

For example,

```
[IBM][CLI Driver]" CLIxxxx: < text >
SQLSTATE=XXXXX {SQLGetData}"
```

0 = do NOT append DB2 CLI function name (default)
1 = append the DB2 CLI function name

This keyword is only useful for debugging.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "SQLError Function (CLI) - Retrieve Error Information" in the *CLI Guide and Reference, Volume 2*
- "SQLGetDiagRec Function (CLI) - Get Multiple Fields Settings of Diagnostic Record" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

ASYNCENABLE CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

AUTOCOMMIT CLI/ODBC Configuration Keyword

Keyword Description:
Specify whether the application commits each statement by default.

db2cli.ini Keyword Syntax:
AUTOCOMMIT = **1** | **0**

Default Setting:
Each statement is treated as a single, complete transaction.

Equivalent Connection Attribute:
SQL_ATTR_AUTOCOMMIT

Usage Notes:

To be consistent with ODBC, DB2 CLI defaults with AUTOCOMMIT on, which means each statement is treated as a single, complete transaction. This keyword can provide an alternative default, but will only be used if the application does not specify a value for SQL_ATTR_AUTOCOMMIT.

1 = SQL_ATTR_AUTOCOMMIT_ON (default)
0 = SQL_ATTR_AUTOCOMMIT_OFF

Note: Most ODBC applications assume the default of AUTOCOMMIT to be on. Extreme care must be used when overriding this default during runtime as the application may depend on this default to operate properly.

This keyword also allows you to specify whether autocommit should be enabled in a Distributed Unit of Work (DUOW) environment. If a connection is part of a coordinated Distributed Unit of Work, and AUTOCOMMIT is not set, the default does not apply; implicit commits arising from autocommit processing are suppressed. If AUTOCOMMIT is set to 1, and the connection is part of a coordinated Distributed Unit of Work, the implicit commits are processed. This may result in severe performance degradation, and possibly other unexpected results elsewhere in the DUOW system. However, some applications may not work at all unless this is enabled.

A thorough understanding of the transaction processing of an application is necessary, especially applications written by a third party, before applying it to a DUOW environment.

Related concepts:

- “Multisite Updates (Two Phase Commit) in CLI Applications” on page 157
- “db2cli.ini Initialization File” on page 293

Related reference:

- “Connection Attributes (CLI) List” in the *CLI Guide and Reference, Volume 2*
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296

BITDATA CLI/ODBC Configuration Keyword

Keyword Description:
Specify whether binary data types are reported as binary or character data types.

db2cli.ini Keyword Syntax:
BITDATA = **1** | **0**

Default Setting:
Report `FOR BIT DATA` and BLOB data types as binary data types.

Usage Notes:

This option allows you to specify whether ODBC binary data types (SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY, and SQL_BLOB), are reported as binary type data. IBM DBMSs support columns with binary data types by defining CHAR, VARCHAR, and LONG VARCHAR columns with the `FOR BIT DATA` attribute. DB2 Universal Database will also support binary data via the BLOB data type (in this case it is mapped to a CLOB data type).

Only set BITDATA = 0 if you are sure that all columns defined as `FOR BIT DATA` or BLOB contain only character data, and the application is incapable of displaying binary data columns.
1 = report `FOR BIT DATA` and BLOB data types as binary data types (default).
0 = report `FOR BIT DATA` and BLOB data types as character data types.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CLIPKG CLI/ODBC Configuration Keyword

Keyword Description:

db2cli.ini Keyword Syntax:
CLIPKG = **3** | **4** | ... | **30**

Default Setting:
Three large packages are generated.

Usage Notes:

This keyword is used to increase the number of sections for SQL statements in CLI/ODBC applications. If it is used, the administrator should explicitly bind the required bind files with the CLIPKG bind option. Also, the `db2cli.ini` file on the server (DB2 UDB Version 6.1 or later on UNIX or Intel platforms) must be updated with the same value of CLIPKG for CLI/JDBC stored procedures.

If the value is NOT an integer between 3 and 30, the default will be used without error or warning.

This setting only applies to large packages (containing 384 sections). The number of small packages (containing 64 sections) is 3 and cannot be changed.

It is recommended that you only increase the number of sections enough to run your application as the packages take up space in the database.

Related concepts:
- "Handles in CLI" on page 19
- "db2cli.ini Initialization File" on page 293

Related reference:
- "BIND" in the *Command Reference*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CLISCHEMA CLI/ODBC Configuration Keyword

Keyword Description:
Set the DB2 ODBC catalog view to use.

db2cli.ini Keyword Syntax:
CLISCHEMA = *ODBC catalog view*

Default Setting:
None - No ODBC catalog view is used

Equivalent Connection Attribute:
SQL_ATTR_CLISCHEMA

Usage Notes:

The DB2 ODBC catalog is designed to improve the performance of schema calls for lists of tables in ODBC applications that connect to host DBMSs through DB2 Connect.

The DB2 ODBC catalog, created and maintained on the host DBMS, contains rows representing objects defined in the real DB2 catalog, but these rows include only the columns necessary to support ODBC operations. The tables in the DB2 ODBC catalog are pre-joined and specifically indexed to support fast catalog access for ODBC applications.

System administrators can create multiple DB2 ODBC catalog views, each containing only the rows that are needed by a particular user group. Each end user can then select the DB2 ODBC catalog view they wish to use (by setting this keyword).

Use of the CLISCHEMA setting is completely transparent to the ODBC application; you can use this option with any ODBC application.

While this keyword has some similar effects as the SYSSCHEMA keyword, CLISCHEMA should be used instead (where applicable).

CLISCHEMA improves data access efficiency: the user-defined tables used with SYSSCHEMA are mirror images of the DB2 catalog tables, and the CLI/ODBC driver still needs to join rows from multiple tables to produce the information required by the CLI/ODBC application. Using CLISCHEMA also results in less lock contention on the catalog tables.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "SYSSCHEMA CLI/ODBC Configuration Keyword" on page 337

CONNECTCODEPAGE CLI/ODBC Configuration Keyword

Keyword Description:
Specifies a specific code page to use when connecting to the data source to avoid extra connection overhead.

db2cli.ini Keyword Syntax:
CONNECTCODEPAGE = **0** | **1** | <any valid DB2 code page>

Default Setting:
Default client code page will be used.

Usage Notes:

Non-Unicode applications always connect to the database using the application's local code page, or the DB2CODEPAGE environment setting. By default, when connecting to DB2 for Unix or Windows, CLI will ensure that Unicode applications will connect to Unicode databases using UTF-8 and UCS-2 code pages, and will connect to non-Unicode databases using the database’s code page. This ensures that there is no unnecessary data loss due to code page conversion. However, there is sometimes a cost incurred while determining the database’s code page.

This keyword allows the user to specify the database’s code page when connecting to a non-Unicode database in order to avoid any extra overhead on the connection.

Specify a value of 1 to cause `SQLDriverConnect()` to return the correct value in the output connection string, so the value can be used on future `SQLDriverConnect()` calls.

Related concepts:
- “db2cli.ini Initialization File” on page 293
- “Unicode CLI Applications” on page 167

Related reference:
- “SQLDriverConnect Function (CLI) - (Expanded) Connect to a Data Source” in the *CLI Guide and Reference, Volume 2*
- “General registry variables” in the *Administration Guide: Performance*
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296

CONNECTNODE CLI/ODBC Configuration Keyword

Keyword Description:
Specify the database partition server to which a connection is to be made.

db2cli.ini Keyword Syntax:
CONNECTNODE = **integer value from 0 to 999** | **SQL_CONN_CATALOG_NODE**

Default Setting:
Database partition server which is defined with port 0 on the machine is used.

Only Applicable when:
Connecting to a partitioned database environment.

Equivalent Connection Attribute:
SQL_ATTR_CONNECT_NODE

Usage Notes:

Used to specify the target database partition server that you want to connect to. This keyword (or attribute setting) overrides the value of the environment variable DB2NODE. Can be set to:

- an integer between 0 and 999
- SQL_CONN_CATALOG_NODE

If this variable is not set, the target defaults to the database partition server that is defined with port 0 on the machine.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related tasks:

- "Setting environment variables on Windows" in the *Administration Guide: Implementation*

Related reference:

- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CONNECTYPE CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CURRENTFUNCTIONPATH CLI/ODBC Configuration Keyword

Keyword Description:
Specify the schema used to resolve function references and data type references in dynamic SQL statements.

db2cli.ini Keyword Syntax:
CURRENTFUNCTIONPATH = *current_function_path*

Default Setting:
See description below.

Usage Notes:

This keyword defines the path used to resolve function references and data type references that are used in dynamic SQL statements. It contains a list of one or more schema-names, where schema-names are enclosed in double quotes and separated by commas.

The default value is ″SYSIBM″,″SYSFUN″,X where X is the value of the USER special register delimited by double quotes. The schema SYSIBM does not need to be specified. If it is not included in the function path, then it is implicitly assumed as the first schema.

This keyword is used as part of the process for resolving unqualified function and stored procedure references that may have been defined in a schema name other than the current user's schema. The order of the schema names determines the order in which the function and procedure names will be resolved.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CURRENTPACKAGESET CLI/ODBC Configuration Keyword

Keyword Description:
Issue ′SET CURRENT PACKAGESET schema′ after every connection.

db2cli.ini Keyword Syntax:
CURRENTPACKAGESET = *schema name*

Default Setting:
The clause is not appended.

Equivalent Connection Attribute:
SQL_ATTR_CURRENT_PACKAGE_SET

Usage Notes:

This option will issue the command ″SET CURRENT PACKAGESET schema″ after every connection to a database. By default this clause is not appended.

This statement sets the schema name (collection identifier) that will be used to select the package to use for subsequent SQL statements.

CLI/ODBC applications issue dynamic SQL statements. Using this option you can control the privileges used to run these statements:
- Choose a schema to use when running SQL statements from CLI/ODBC applications.
- Ensure the objects in the schema have the desired privileges and then rebind accordingly.
- Set the CURRENTPACKAGESET option to this schema.

The SQL statements from the CLI/ODBC applications will now run under the specified schema and use the privileges defined there.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "SET CURRENT PACKAGESET statement" in the *SQL Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CURRENTSCHEMA CLI/ODBC Configuration Keyword

Keyword Description:
Specify the schema used in a SET CURRENT SCHEMA statement upon a successful connection.

db2cli.ini Keyword Syntax:
CURRENTSCHEMA = *schema name*

Default Setting:
No statement is issued.

Usage Notes:

Upon a successful connect, if this option is set, a SET CURRENT SCHEMA statement is sent to the DBMS. This allows the end user or application to name SQL objects without having to qualify them by schema name.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "SET SCHEMA statement" in the *SQL Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CURRENTSQLID CLI/ODBC Configuration Keyword

Keyword Description:
Specify the ID used in a SET CURRENT SQLID statement sent to the DBMS upon a successful connection.

db2cli.ini Keyword Syntax:
CURRENTSQLID = *current_sqlid*

Default Setting:
No statement is issued.

Only Applicable when:
connecting to those DB2 DBMS's where SET CURRENT SQLID is supported.

Usage Notes:

Upon a successful connection, if this option is set, a SET CURRENT SQLID statement is sent to the DBMS. This allows the end user and the application to name SQL objects without having to qualify them by schema name.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

CURSORHOLD CLI/ODBC Configuration Keyword

Keyword Description:
Effect of a transaction completion on open cursors.

db2cli.ini Keyword Syntax:
CURSORHOLD = **$\underline{1}$** | **0**

Default Setting:
Selected--Cursors are not destroyed.

Equivalent Statement Attribute:
SQL_ATTR_CURSOR_HOLD

Usage Notes:

This option controls the effect of a transaction completion on open cursors.
1 = SQL_CURSOR_HOLD_ON, the cursors are not destroyed when the transaction is committed (default).

0 = SQL_CURSOR_HOLD_OFF, the cursors are destroyed when the transaction is committed.

Note: Cursors are always closed when transactions are rolled back.

This option affects the result returned by `SQLGetInfo()` when called with SQL_CURSOR_COMMIT_BEHAVIOR or SQL_CURSOR_ROLLBACK_BEHAVIOR. The value of CURSORHOLD is ignored if connecting to DB2 for VSE & VM where cursor `with hold` is not supported.

You can use this option to tune performance. It can be set to SQL_CURSOR_HOLD_OFF (0) if you are sure that your application:
1. Does not have behavior that is dependent on the SQL_CURSOR_COMMIT_BEHAVIOR or the SQL_CURSOR_ROLLBACK_BEHAVIOR information returned via `SQLGetInfo()`, and
2. Does not require cursors to be preserved from one transaction to the next.

The DBMS will operate more efficiently with CURSORHOLD disabled, as resources no longer need to be maintained after the end of a transaction.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "SQLGetInfo Function (CLI) - Get General Information" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DATABASE CLI/ODBC Configuration Keyword

Keyword Description:
Database on the server to connect to when using a File DSN.

db2cli.ini Keyword Syntax:
DATABASE = *database name*

Default Setting:
None

Only Applicable when:
PROTOCOL set to TCPIP

Usage Notes:

When using a File DSN you must use this option to specify the database on the server to connect to. This value has nothing to do with any database alias name specified on the client, it must be set to the database name on the server itself.

This setting is only considered when the PROTOCOL option is set to TCPIP.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "HOSTNAME CLI/ODBC Configuration Keyword" on page 320
- "PROTOCOL CLI/ODBC Configuration Keyword" on page 328
- "SERVICENAME CLI/ODBC Configuration Keyword" on page 331

DB2DEGREE CLI/ODBC Configuration Keyword

Keyword Description:
Set the degree of parallelism for the execution of SQL statements.

db2cli.ini Keyword Syntax:
DB2DEGREE = <u>**0**</u> | **integer value from 1 to 32767** | **ANY**

Default Setting:
No SET CURRENT DEGREE statement is issued.

Only Applicable when:
connecting to a cluster database system.

Usage Notes:

This option only applies to a DB2 Version 5 or later server. If the value specified is anything other than 0 (the default) then DB2 CLI will issue the following SQL statement after a successful connection:

```
SET CURRENT DEGREE value
```

This specifies the degree of parallelism for the execution of the SQL statements. The database manager will determine the degree of parallelism if you specify ANY.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:

- “SET CURRENT DEGREE statement” in the *SQL Reference, Volume 2*
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296

DB2EXPLAIN CLI/ODBC Configuration Keyword

Keyword Description:
Determines whether Explain snapshot and/or Explain table information will be generated by the server.

db2cli.ini Keyword Syntax:
DB2EXPLAIN = **<u>0</u>** | **1** | **2** | **3**

Default Setting:
Neither Explain snapshot nor Explain table information will be generated by the server.

Equivalent Connection Attribute:
SQL_ATTR_DB2EXPLAIN

Usage Notes:

This keyword determines whether Explain snapshot and/or Explain table information will be generated by the server.

- 0 = both off (default)

  A ’SET CURRENT EXPLAIN SNAPSHOT=NO’ and a ’SET CURRENT EXPLAIN MODE=NO’ statement will be sent to the server to disable both the Explain snapshot and the Explain table information capture facilities.
- 1 = Only Explain snapshot facility on

  A ’SET CURRENT EXPLAIN SNAPSHOT=YES’ and a ’SET CURRENT EXPLAIN MODE=NO’ statement will be sent to the server to enable the Explain snapshot facility, and disable the Explain table information capture facility.
- 2 = Only Explain table information capture facility on

  A ’SET CURRENT EXPLAIN MODE=YES’ and a ’SET CURRENT EXPLAIN SNAPSHOT=NO’ will be sent to the server to enable the Explain table information capture facility and disable the Explain snapshot facility.
- 3 = Both on

  A ’SET CURRENT EXPLAIN MODE=YES’ and a ’SET CURRENT EXPLAIN SNAPSHOT=YES’ will be sent to the server to enable both the Explain snapshot and the Explain table information capture facilities.

Explain information is inserted into Explain tables, which must be created before the Explain information can be generated. The current authorization ID must have INSERT privilege for the Explain tables.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI and ODBC Function Summary" in the *CLI Guide and Reference, Volume 2*
- "EXPLAIN statement" in the *SQL Reference, Volume 2*
- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*

DB2OPTIMIZATION CLI/ODBC Configuration Keyword

Keyword Description:
Set the query optimization level.

db2cli.ini Keyword Syntax:
DB2OPTIMIZATION = *integer value from 0 to 9*

Default Setting:
No SET CURRENT QUERY OPTIMIZATION statement issued.

Usage Notes:

If this option is set then DB2 CLI will issue the following SQL statement after a successful connection:

```
SET CURRENT QUERY OPTIMIZATION positive number
```

This specifies the query optimization level at which the optimizer should operate the SQL queries.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "SET CURRENT QUERY OPTIMIZATION statement" in the *SQL Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DBALIAS CLI/ODBC Configuration Keyword

Keyword Description:
Specify the database alias for a Data Source Name greater than 8 characters.

db2cli.ini Keyword Syntax:
DBALIAS = *dbalias*

Default Setting:
Use the DB2 database alias as the ODBC Data Source Name.

Usage Notes:

This keyword allows for Data Source Names of greater than 8 single byte characters. The Data Source Name (DSN) is the name, enclosed in square brackets, that denotes the section header in the `db2cli.ini`. Typically, this section header is the database alias name which has a maximum length of 8 bytes. A user who wishes to refer to the data source with a longer, more meaningful name, can place the longer name in the section header, and set this keyword value to the database alias used on the CATALOG command. Here is an example:

```
; The much longer name maps to an 8 single byte character dbalias
[MyMeaningfulName]
DBALIAS=DB2DBT10
```

The end user can specify `[MyMeaningfulName]` as the name of the data source on connect while the actual database alias is `DB2DBT10`.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CATALOG DATABASE" in the *Command Reference*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DBNAME CLI/ODBC Configuration Keyword

Keyword Description:
Specify the database name to reduce the time it takes for the application to query z/OS or OS/390 table information.

db2cli.ini Keyword Syntax:
DBNAME = *dbname*

Default Setting:
Do not filter on the DBNAME column.

Only Applicable when:
connecting to DB2 for z/OS and OS/390.

Usage Notes:

This option is only used when connecting to DB2 for z/OS and OS/390, and only if (*base*) table catalog information is requested by the application. If a large number of tables exist in the z/OS or OS/390 subsystem, a *dbname* can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application.

If this option is set then the statement IN DATABASE *dbname* will be appended to various statements such as CREATE TABLE.

This value maps to the `DBNAME` column in the z/OS or OS/390 system catalog tables. If no value is specified, or if views, synonyms, system tables, or aliases are also specified via TABLETYPE, only table information will be restricted; views, aliases, and synonyms are not restricted with DBNAME. It can be used in conjunction with SCHEMALIST, and TABLETYPE to further limit the number of tables for which information will be returned.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "SCHEMALIST CLI/ODBC Configuration Keyword" on page 330
- "TABLETYPE CLI/ODBC Configuration Keyword" on page 337

DEFAULTPROCLIBRARY CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DEFERREDPREPARE CLI/ODBC Configuration Keyword

Keyword Description:
Minimize network flow by combining the PREPARE request with the corresponding execute request.

db2cli.ini Keyword Syntax:
DEFERREDPREPARE = **0** | **1**

Default Setting:
The prepare request will be delayed until the execute request is sent.

Equivalent Statement Attribute:
SQL_ATTR_DEFERRED_PREPARE

Usage Notes:

Defers sending the PREPARE request until the corresponding execute request is issued. The two requests are then combined into one command/reply flow (instead of two) to minimize network flow and to improve performance.

- 0 = SQL_DEFERRED_PREPARE_OFF. The PREPARE request will be executed the moment it is issued.
- 1 = SQL_DEFERRED_PREPARE_ON (default). Defer the execution of the PREPARE request until the corresponding execute request is issued.

  If the target DBMS does not support deferred prepare, the client disables deferred prepare for that connection.

Note: When deferred prepare is enabled, the row and cost estimates normally returned in the SQLERRD(3) and SQLERRD(4) of the SQLCA of a PREPARE statement may become zeros. This may be of concern to users who want to use these values to decide whether or not to continue the SQL statement.

Related concepts:
- "Deferred Prepare in CLI Applications" on page 32
- "db2cli.ini Initialization File" on page 293

Related tasks:
- "Preparing and Executing SQL Statements in CLI Applications" on page 31

Related reference:
- "PREPARE statement" in the *SQL Reference, Volume 2*
- "SQLCA (SQL communications area)" in the *SQL Reference, Volume 1*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DISABLEKEYSETCURSOR CLI/ODBC Configuration Keyword

Keyword Description:
Disables keyset-driven scrollable cursors.

db2cli.ini Keyword Syntax:
DISABLEKEYSETCURSOR = **0** | **1**

Default Setting:
Keyset-driven scrollable cursors are returned when requested.

Usage Notes:

When set to 1, this keyword forces the CLI driver to return a static cursor to the application, even if the application has requested a keyset-driven scrollable cursor. The default setting (0) causes keyset-driven cursors to be returned when the application requests them. This keyword can be used to restore behavior before scrollable cursors were supported.

Related concepts:
- "Cursors in CLI Applications" on page 75
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DISABLEMULTITHREAD CLI/ODBC Configuration Keyword

Keyword Description:
Disable multithreading.

db2cli.ini Keyword Syntax:
DISABLEMULTITHREAD = **0** | **1**

Default Setting:
Multithreading is enabled.

Usage Notes:

The CLI/ODBC driver is capable of supporting multiple concurrent threads.

This option is used to enable or disable multi-thread support.
- 0 = Multithreading is enabled (default).
- 1 = Disable Multithreading.

If multithreading is disabled then all calls for all threads will be serialized at the process level. Use this setting for multithreaded applications that require serialized behavior.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- "Multithreaded CLI Applications" on page 151
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

DISABLEUNICODE CLI/ODBC Configuration Keyword

Keyword Description:
Disable underlying Unicode support.

db2cli.ini Keyword Syntax:
DISABLEUNICODE = **0** | **1**

Default Setting:
Unicode support is enabled.

Usage Notes:

With Unicode support enabled, and when called by a Unicode application, CLI will attempt to connect to the database using the best client code page possible to ensure there is no unnecessary data loss due to code page conversion. This may increase the connection time as code pages are exchanged, or may cause code page conversions on the client that did not occur before this support was added.

Setting this keyword to true (1) will cause all Unicode data to be converted to the application's local code page first, before the data is sent to the server. This can cause data loss for any data that cannot be represented in the local code page.

Related concepts:
- "db2cli.ini Initialization File" on page 293
- "Unicode CLI Applications" on page 167

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

GRANTEELIST CLI/ODBC Configuration Keyword

Keyword Description:
Reduce the amount of information returned when the application gets a list of table or column privileges.

db2cli.ini Keyword Syntax:
GRANTEELIST = *" 'userID1', 'userID2',... 'userIDn' "*

Default Setting:
Do not filter the results.

Usage Notes:

This option can be used to reduce the amount of information returned when the application gets a list of privileges for tables in a database, or columns in a table. The list of authorization IDs specified is used as a filter; the only tables or columns that are returned are those with privileges that have been granted *TO* those IDs.

Set this option to a list of one or more authorization IDs that have been granted privileges, delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
GRANTEELIST=" 'USER1', 'USER2', 'USER8' "
```

In the above example, if the application gets a list of privileges for a specific table, only those columns that have a privilege granted *TO* USER1, USER2, or USER8 would be returned.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "GRANTORLIST CLI/ODBC Configuration Keyword" on page 319

GRANTORLIST CLI/ODBC Configuration Keyword

Keyword Description:
Reduce the amount of information returned when the application gets a list of table or column privileges.

db2cli.ini Keyword Syntax:
GRANTORLIST = *" 'userID1', 'userID2',... 'userIDn' "*

Default Setting:
Do not filter the results.

Usage Notes:

This option can be used to reduce the amount of information returned when the application gets a list of privileges for tables in a database, or columns in a table. The list of authorization IDs specified is used as a filter; the only tables or columns that are returned are those with privileges that have been granted *BY* those IDs.

Set this option to a list of one or more authorization IDs that have granted privileges, delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
   GRANTORLIST=" 'USER1', 'USER2', 'USER8' "
```

In the above example, if the application gets a list of privileges for a specific table, only those columns that have a privilege granted *BY* USER1, USER2, or USER8 would be returned.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "GRANTEELIST CLI/ODBC Configuration Keyword" on page 319

HOSTNAME CLI/ODBC Configuration Keyword

Keyword Description:
The server system's host name or IP address, used with file DSN or in a DSN-less connection.

db2cli.ini Keyword Syntax:
HOSTNAME = *host name* | *IP Address*

Default Setting:
None

Only Applicable when:
PROTOCOL set to TCPIP

Usage Notes:

Use this option in conjunction with the SERVICENAME option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the PROTOCOL option is set to TCPIP.

Specify either the server system's host name or its IP address.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "PROTOCOL CLI/ODBC Configuration Keyword" on page 328
- "SERVICENAME CLI/ODBC Configuration Keyword" on page 331

IGNOREWARNINGS CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "IGNOREWARNLIST CLI/ODBC Configuration Keyword" on page 321

IGNOREWARNLIST CLI/ODBC Configuration Keyword

Keyword Description:
Ignore specified sqlstates.

db2cli.ini Keyword Syntax:
IGNOREWARNLIST = "'sqlstate1', 'sqlstate2', ..."

Default Setting:
Warnings are returned as normal

Usage Notes:

On rare occasions an application may not correctly handle some warning messages, but does not want to ignore all warning messages. This keyword can be used to indicate which warnings are not to be passed on to the application.

If an sqlstate is included in both IGNOREWARNLIST and WARNINGLIST, it will be ignored altogether.

Each sqlstate must be in uppercase, delimited with single quotes and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
IGNOREWARNLIST="'01000', '01004','01504'"
```

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "WARNINGLIST CLI/ODBC Configuration Keyword" on page 353

KEEPCONNECT CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

KEEPSTATEMENT CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

LOBMAXCOLUMNSIZE CLI/ODBC Configuration Keyword

Keyword Description:
Override default COLUMN_SIZE for LOB data types.

db2cli.ini Keyword Syntax:
LOBMAXCOLUMNSIZE = *integer greater than zero*

Default Setting:
2 Gigabytes (1G for DBCLOB)

Only Applicable when:
LONGDATACOMPAT option is used.

Usage Notes:

This will override the 2 Gigabyte (1G for DBCLOB) value that is returned by `SQLGetTypeInfo()` for the COLUMN_SIZE column for SQL_CLOB, SQL_BLOB, and SQL_DBCLOB SQL data types. Subsequent CREATE TABLE statements that contain LOB columns will use the column size value you set here instead of the default.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "LONGDATACOMPAT CLI/ODBC Configuration Keyword" on page 323

LONGDATACOMPAT CLI/ODBC Configuration Keyword

Keyword Description:
Report LOBs as long data types or as large object types.

db2cli.ini Keyword Syntax:
LONGDATACOMPAT = **0** | **1**

Default Setting:
Reference LOB data types as large object types.

Equivalent Connection Attribute:
SQL_ATTR_LONGDATA_COMPAT

Usage Notes:

This option indicates to DB2 CLI what data type the application expects when working with a database with large object (LOB) columns.

Database data type	Large Objects (0--Default)	Long Data Types (1)
CLOB	SQL_CLOB	SQL_LONGVARCHAR
BLOB	SQL_BLOB	SQL_LONGVARBINARY
DBCLOB	SQL_DBCLOB	SQL_LONGVARGRAPHIC

This option is useful when running ODBC applications that cannot handle the large object data types.

The DB2 CLI/ODBC option LOBMAXCOLUMNSIZE can be used in conjunction with this option to reduce the default size declared for the data.

Related concepts:

- "Large Object Usage in CLI Applications" on page 116
- "db2cli.ini Initialization File" on page 293

Related reference:

- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "LOBMAXCOLUMNSIZE CLI/ODBC Configuration Keyword" on page 323

MODE CLI/ODBC Configuration Keyword

Keyword Description:
Default connection mode.

db2cli.ini Keyword Syntax:
MODE = **SHARE** | **EXCLUSIVE**

Default Setting:
SHARE

Not Applicable when:
connecting to a host or iSeries server.

Usage Notes:

Sets the CONNECT mode to either SHARE or EXCLUSIVE. If a mode is set by the application at connect time, this value is ignored. The default is SHARE.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CONNECT (Type 1) statement" in the *SQL Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

OPTIMIZEFORNROWS CLI/ODBC Configuration Keyword

Keyword Description:
Append 'OPTIMIZE FOR n ROWS' clause to every select statement.

db2cli.ini Keyword Syntax:
OPTIMIZEFORNROWS = *integer*

Default Setting:
The clause is not appended.

Equivalent Statement Attribute:
SQL_ATTR_OPTIMIZE_FOR_NROWS

Usage Notes:

This option will append the "OPTIMIZE FOR n ROWS" clause to every select statement, where n is an integer larger than 0. If set to 0 (the default) this clause will not be appended.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "SELECT statement" in the *SQL Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

PATCH1 CLI/ODBC Configuration Keyword

Keyword Description:
Use work-arounds for known CLI/ODBC application problems.

db2cli.ini Keyword Syntax:
PATCH1 = { **0** | **1** | **2** | **4** | **8** | **16** | **...** }

Default Setting:
Use no work-arounds.

Usage Notes:

This keyword is used to specify a work-around for known problems with ODBC applications. The value specified can be for none, one, or multiple work-arounds. The patch values specified here are used in conjunction with any PATCH2 values that may also be set.

Using the DB2 CLI/ODBC Settings notebook you can select one or more patches to use. If you set the values in the `db2cli.ini` file itself and want to use multiple patch values then simply add the values together to form the keyword value. For example, if you want the patches 1, 4, and 8, then specify PATCH1=13.

- 0 = No work around (default)

The DB2 CLI/ODBC Settings notebook has a list of values. (This information is also contained in the `README` file if there are any current patch values for that platform).

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "PATCH2 CLI/ODBC Configuration Keyword" on page 326

PATCH2 CLI/ODBC Configuration Keyword

Keyword Description:
Use work-arounds for known CLI/ODBC application problems.

db2cli.ini Keyword Syntax:
PATCH2 = "*patch value 1, patch value 2, patch value 3,* ..."

Default Setting:
Use no work-arounds

Usage Notes:

This keyword is used to specify a work-around for known problems with CLI/ODBC applications. The value specified can be for none, one, or multiple work-arounds. The patch values specified here may be used in conjunction with any PATCH1 values that may also be set.

When specifying multiple patches, the values are specified in a comma delimited string (unlike the PATCH1 option where the values are added together and the sum is used).
- 0 = No work around (default)

To set PATCH2 values 3, 4 and 8 you would specify:

```
PATCH2="3, 4, 8"
```

The PATCH2 values are contained in the README file if there are any current patch values for that platform.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "PATCH1 CLI/ODBC Configuration Keyword" on page 325

PORT CLI/ODBC Configuration Keyword

Keyword Description:
The server system's service name or port number, used with a file DSN or in a DSN-less connection.

db2cli.ini Keyword Syntax:
PORT = *service name* | *port number*

Default Setting:
None

Only Applicable when:
PROTOCOL set to TCPIP

Usage Notes:

Use this option in conjunction with the HOSTNAME option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the PROTOCOL option is set to TCPIP.

Specify either the server system's service name or its port number. The service name must be available for lookup at the client machine if it is used.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “HOSTNAME CLI/ODBC Configuration Keyword” on page 320
- “PROTOCOL CLI/ODBC Configuration Keyword” on page 328
- “SERVICENAME CLI/ODBC Configuration Keyword” on page 331

PROTOCOL CLI/ODBC Configuration Keyword

Keyword Description:
Communications protocol used for File DSN or in a DSN-less connection.

db2cli.ini Keyword Syntax:
PROTOCOL = **TCPIP**

Default Setting:
none

Usage Notes:

TCP/IP is the only protocol supported when using a File DSN. Set the option to the string TCPIP (without the slash).

When this option is set then the following options must also be set:
- DATABASE
- SERVICENAME
- HOSTNAME

Related concepts:
- “db2cli.ini Initialization File” on page 293

Related reference:
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “DATABASE CLI/ODBC Configuration Keyword” on page 310
- “HOSTNAME CLI/ODBC Configuration Keyword” on page 320
- “SERVICENAME CLI/ODBC Configuration Keyword” on page 331

PWD CLI/ODBC Configuration Keyword

Keyword Description:
Define default password.

db2cli.ini Keyword Syntax:
PWD = *password*

Default Setting:
None

Usage Notes:

This *password* value is used if a password is not provided by the application at connect time.

It is stored as plain text in the db2cli.ini file and is therefore not secure.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "UID CLI/ODBC Configuration Keyword" on page 351

QUERYTIMEOUTINTERVAL CLI/ODBC Configuration Keyword

Keyword Description:
Delay (in seconds) between checks for a query timeout.

db2cli.ini Keyword Syntax:
QUERYTIMEOUTINTERVAL = **0** | **5** | **positive integer**

Default Setting:
5 seconds

Usage Notes:

An application can use the `SQLSetStmtAttr()` function to set the SQL_ATTR_QUERY_TIMEOUT statement attribute. This attribute indicates the number of seconds to wait for an SQL statement to complete executing before attempting to cancel the execution and returning to the application.

The QUERYTIMEOUTINTERVAL configuration keyword is used to indicate how long the CLI driver should wait between checks to see if the query has completed.

For instance, suppose SQL_ATTR_QUERY_TIMEOUT is set to 25 seconds (timeout after waiting for 25 seconds), and QUERYTIMEOUTINTERVAL is set to 10 seconds (check the query every 10 seconds). The query will not time out until 30 seconds (the first check AFTER the 25 second limit).

Note: DB2 CLI implements query timeout by starting a thread that periodically queries the status of each executing query. The QUERYTIMEOUTINTERVAL value specifies how long the query timeout thread waits between checks for expired queries. Because this is an asynchronous operation to the queries being executed, it is possible that a given query may not be timed out until SQL_ATTR_QUERY_TIMEOUT + QUERYTIMEOUTINTERVAL seconds. In the example above, the best-case timeout would be at 26 seconds, and the worst-case timeout would be at 35 seconds.

There may be cases where the SQL_ATTR_QUERY_TIMEOUT is set to a value which is too low, and the query should NOT be timed-out. If the application cannot be modified (i.e., a third party ODBC application), then the QUERYTIMEOUTINTERVAL can be set to 0, and the CLI driver will ignore the SQL_ATTR_QUERY_TIMEOUT setting, and therefore wait for SQL statements to complete execution before returning to the application.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Alternatively, QUERYTIMEOUTINTERVAL can be set to a value that is larger than the SQL_ATTR_QUERY_TIMEOUT setting, thus preventing timeouts from occurring at the specified interval. For example, if the application sets a 15 second SQL_ATTR_QUERY_TIMEOUT value, but the server requires at least 30 seconds to execute the query, the QUERYTIMEOUTINTERVAL can be set to a value of 30 seconds or so to prevent this query from timing out after 15 seconds.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "SQLSetStmtAttr Function (CLI) - Set Options Related to a Statement" in the *CLI Guide and Reference, Volume 2*
- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

SCHEMALIST CLI/ODBC Configuration Keyword

Keyword Description:
Restrict schemas used to query table information.

db2cli.ini Keyword Syntax:
SCHEMALIST = *" 'schema1', 'schema2',... 'schemaN' "*

Default Setting:
None

Usage Notes:

SCHEMALIST is used to provide a more restrictive default, and therefore improve performance, for those applications that list every table in the DBMS.

If there are a large number of tables defined in the database, a schema list can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application. Each schema name is case-sensitive, must be delimited with single quotes, and separated by commas. The entire string must also be enclosed in double quotes. For example:

```
SCHEMALIST="'USER1','USER2','USER3'"
```

For DB2 for z/OS, CURRENT SQLID can also be included in this list, but without the single quotes, for example:

```
SCHEMALIST="'USER1',CURRENT SQLID,'USER3'"
```

The maximum length of the string is 256 characters.

This option can be used in conjunction with DBNAME and TABLETYPE to further limit the number of tables for which information will be returned.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "DBNAME CLI/ODBC Configuration Keyword" on page 314
- "HOSTNAME CLI/ODBC Configuration Keyword" on page 320

SERVICENAME CLI/ODBC Configuration Keyword

Keyword Description:
The server system's service name or port number, used with file DSN or in a DSN-less connection.

db2cli.ini Keyword Syntax:
SERVICENAME = *service name* | *port number*

Default Setting:
None

Only Applicable when:
PROTOCOL set to TCPIP

Usage Notes:

Use this option in conjunction with the HOSTNAME option to specify the required attributes for a TCP/IP connection from this client machine to a server running DB2. These two values are only considered when the PROTOCOL option is set to TCPIP.

Specify either the server system's service name or its port number. The service name must be available for lookup at the client machine if it is used.

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "HOSTNAME CLI/ODBC Configuration Keyword" on page 320
- "PROTOCOL CLI/ODBC Configuration Keyword" on page 328

SKIPTRACE CLI/ODBC Configuration Keyword

Keyword Description:
Excludes CLI trace information from the DB2 trace.

db2cli.ini Keyword Syntax:
SKIPTRACE = **0** | **1**

Default Setting:
Do not skip the trace function.

Usage Notes:

This keyword can improve performance by allowing the DB2 trace function to bypass CLI applications. Therefore, if the DB2 trace facility db2trc is turned on and this keyword is set to 1, the trace will not contain information from the execution of the CLI application.

Turning SKIPTRACE on is recommended for production environments on the UNIX platform where trace information is not required. Test environments may benefit, however, from having trace output, so this keyword can be turned off (or left at its default setting) when detailed execution information is desired.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- "db2cli.ini Initialization File" on page 293

Related reference:
- "db2trc - Trace" in the *Command Reference*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

STATICCAPFILE CLI/ODBC Configuration Keyword

Keyword Description:
Specify the Capture File name and optionally the path where it will be saved.

db2cli.ini Keyword Syntax:
STATICCAPFILE = < **Full file name** >

Default Setting:
None - you must specify a capture file.

Only Applicable when:
STATICMODE is set to Capture or Match

Usage Notes:

This keyword is used to specify the Capture File name and optionally the directory where it will be saved.

Related concepts:
- "Capture File for CLI/ODBC/JDBC Static Profiling" on page 220
- "db2cli.ini Initialization File" on page 293

Related tasks:
- "Creating Static SQL with CLI/ODBC/JDBC Static Profiling" on page 217

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

- “STATICLOGFILE CLI/ODBC Configuration Keyword” on page 334
- “STATICMODE CLI/ODBC Configuration Keyword” on page 334
- “STATICPACKAGE CLI/ODBC Configuration Keyword” on page 335

STATICLOGFILE CLI/ODBC Configuration Keyword

Keyword Description:
Specify the Static Profiling Log File name and optionally the directory where it will be saved.

db2cli.ini Keyword Syntax:
STATICLOGFILE = < **Full file name** >

Default Setting:
No Static Profiling Log is created. If a filename is specified without a pathname then the current path will be used.

Only Applicable when:
STATICMODE is set to Capture or Match

Usage Notes:

This keyword is used to specify the Static Profiling Log File name and optionally the directory where it will be saved.

Related concepts:
- “db2cli.ini Initialization File” on page 293

Related tasks:
- “Creating Static SQL with CLI/ODBC/JDBC Static Profiling” on page 217

Related reference:
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “STATICCAPFILE CLI/ODBC Configuration Keyword” on page 333
- “STATICMODE CLI/ODBC Configuration Keyword” on page 334
- “STATICPACKAGE CLI/ODBC Configuration Keyword” on page 335

STATICMODE CLI/ODBC Configuration Keyword

Keyword Description:
Specify whether the CLI/ODBC application will capture SQL or use a static SQL Package for this DSN.

db2cli.ini Keyword Syntax:
STATICMODE = **DISABLED** | **CAPTURE** | **MATCH**

Default Setting:
Disabled - SQL statements are not captured and no static SQL package is used.

Usage Notes:

This option allows you to specify how the SQL issued by the CLI/ODBC application for this DSN will be processed:

- DISABLED = Static mode disabled. No special processing. The CLI/ODBC statements will be executed as dynamic SQL with no change. This is the default.
- CAPTURE = Capture Mode. Execute the CLI/ODBC statements as dynamic SQL. If the SQL statements are successful, they will be captured into a file (known as the Capture File) to be bound by the DB2CAP command later.
- MATCH = Match mode. Execute the CLI/ODBC statements as static SQL statements if a matching statement is found in the Capture Files specified in STATICPACKAGE. The Capture File must first be bound by the DB2CAP command.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related tasks:

- "Creating Static SQL with CLI/ODBC/JDBC Static Profiling" on page 217

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "STATICCAPFILE CLI/ODBC Configuration Keyword" on page 333
- "STATICLOGFILE CLI/ODBC Configuration Keyword" on page 334
- "STATICPACKAGE CLI/ODBC Configuration Keyword" on page 335

STATICPACKAGE CLI/ODBC Configuration Keyword

Keyword Description:
Specify the package to be used with the static profiling feature.

db2cli.ini Keyword Syntax:
STATICPACKAGE = *collection_id.package_name*

Default Setting:
None - you must specify a package name.

Only Applicable when:
STATICMODE is set to CAPTURE

Usage Notes:

This keyword is used to specify the package to be used when the application runs in Match Mode. You first need to use Capture Mode to create the Capture File.

Only the first 7 characters of the indicated package name will be used. A one-byte suffix will be added to represent each isolation level, as follows:

- 0 for Uncommitted Read (UR)
- 1 for Cursor Stability (CS)
- 2 for Read Stability (RS)
- 3 for Repeatable Read (RR)
- 4 for No Commit (NC)

Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*
- "db2cli.ini Initialization File" on page 293

Related tasks:

- "Creating Static SQL with CLI/ODBC/JDBC Static Profiling" on page 217

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "STATICCAPFILE CLI/ODBC Configuration Keyword" on page 333
- "STATICLOGFILE CLI/ODBC Configuration Keyword" on page 334
- "STATICMODE CLI/ODBC Configuration Keyword" on page 334

SYNCPOINT CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

SYSSCHEMA CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

TABLETYPE CLI/ODBC Configuration Keyword

Keyword Description:
Define a default list of TABLETYPES returned when querying table information.

db2cli.ini Keyword Syntax:
TABLETYPE = " *'TABLE'* | *,'ALIAS'* | *,'VIEW'* | *, 'INOPERATIVE VIEW'* | *, 'SYSTEM TABLE'* | *,'SYNONYM'* "

Default Setting:
No default list of TABLETYPES is defined.

Usage Notes:

If there is a large number of tables defined in the database, a tabletype string can be specified to reduce the time it takes for the application to query table information, and reduce the number of tables listed by the application.

Any number of the values can be specified. Each type must be delimited with single quotes, separated by commas, and in uppercase. The entire string must also be enclosed in double quotes. For example:

```
TABLETYPE="'TABLE','VIEW'"
```

This option can be used in conjunction with DBNAME and SCHEMALIST to further limit the number of tables for which information will be returned.

TABLETYPE is used to provide a default for the DB2 CLI function that retrieves the list of tables, views, aliases, and synonyms in the database. If the application does not specify a table type on the function call, and this keyword is not used, information about all table types is returned. If the application does supply a value for the *tabletype* on the function call, then that argument value will override this keyword value.

If TABLETYPE includes any value other than TABLE, then the DBNAME keyword setting cannot be used to restrict information to a particular DB2 for z/OS database.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "DBNAME CLI/ODBC Configuration Keyword" on page 314
- "SCHEMALIST CLI/ODBC Configuration Keyword" on page 330

TEMPDIR CLI/ODBC Configuration Keyword

Keyword Description:
Define the directory used for temporary files.

db2cli.ini Keyword Syntax:
TEMPDIR = < *full path name* >

Default Setting:
Use the system temporary directory specified by the TEMP or TMP environment variables.

Usage Notes:

When working with Large Objects (CLOBS, BLOBS, etc...), when data conversion occurs, or when data is sent to the server in pieces, a temporary file is often created on the client machine to store the information. Using this option you can specify a location for these temporary files. The system temporary directory will be used if nothing is specified.

The keyword is placed in the data source specific section of the db2cli.ini file, and has the following syntax:

- TempDir= F:\DB2TEMP

The path specified must already exist and the user executing the application must have the appropriate authorities to write files to it. When the DB2 CLI Driver attempts to create temporary files, an SQLSTATE of HY507 will be returned if the path name is invalid, or if the temporary files cannot be created in the directory specified.

Related concepts:

- "Large Object Usage in CLI Applications" on page 116
- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

TRACE CLI/ODBC Configuration Keyword

Keyword Description:
Turn on the DB2 CLI/ODBC trace facility.

db2cli.ini Keyword Syntax:
TRACE = **0** | **1**

Default Setting:
No trace information is captured.

Equivalent Connection Attribute:
SQL_ATTR_TRACE

Usage Notes:

When this option is on (1), CLI/ODBC trace records are appended to the file indicated by the TRACEFILENAME configuration parameter or to files in the subdirectory indicated by the TRACEPATHNAME configuration parameter. TRACE will have no effect if neither TRACEFILENAME or TRACEPATHNAME is set.

The TRACEREFRESHINTERVAL keyword sets the interval in seconds at which the TRACE keyword is read from the db2cli.ini file. This allows you to dynamically turn off the CLI/ODBC trace within n seconds.

For example, to set up a CLI/ODBC trace file that is written to disk after each trace entry:

```
[COMMON]
TRACE=1
TRACEFILENAME=E:\TRACES\CLI\MONDAY.CLI
TRACEFLUSH=1
```

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:

- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343
- "TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword" on page 346

TRACECOMM CLI/ODBC Configuration Keyword

Keyword Description:
Include information about each network request in the trace file.

db2cli.ini Keyword Syntax:
TRACECOMM = **0** | **1**

Default Setting:
0 - No network request information is captured.

Only Applicable when:
the CLI/ODBC TRACE option is turned on.

Usage Notes:

When TRACECOMM is set on (1) then information about each network request will be included in the trace file.

This option is only used when the TRACE CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343

TRACEFILENAME CLI/ODBC Configuration Keyword

Keyword Description:
File used to store the DB2 CLI/ODBC trace information.

db2cli.ini Keyword Syntax:
TRACEFILENAME = < **Full file name** >

Default Setting:
None

Only Applicable when:
the TRACE option is turned on.

Equivalent Connection Attribute:
SQL_ATTR_TRACEFILE

Usage Notes:

If the file specified does not exist, then it will be created; otherwise, the new trace information will be appended to the end of the file. However, the path the file is expected in must exist.

If the filename given is invalid or if the file cannot be created or written to, no trace will occur and no error message will be returned.

This option is only used when the TRACE option is turned on. This will be done automatically when you set this option in the CLI/ODBC Configuration utility.

The TRACEPATHNAME option will be ignored if this option is set.

DB2 CLI trace should only be used for debugging purposes. It will slow down the execution of the CLI/ODBC driver, and the trace information can grow quite large if it is left on for extended periods of time.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:
- "Connection Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343

TRACEFLUSH CLI/ODBC Configuration Keyword

Keyword Description:
Force a write to disk after n CLI/ODBC trace entries.

db2cli.ini Keyword Syntax:
TRACEFLUSH = 0 | **positive integer**

Default Setting:
Do not write after every entry.

Only Applicable when:
the CLI/ODBC TRACE option is turned on.

Usage Notes:

Set this option to a positive integer to force the DB2 CLI driver to close and re-open the appropriate trace file after the specified number of trace entries. The smaller the value of the TRACEFLUSH keyword, the greater the impact DB2 CLI tracing has on the performance of the application. If TRACEFLUSH is left at its default value (0), the trace file will only be closed when the application terminates normally. If the application process terminates abnormally, valuable trace information may be lost. Setting TRACEFLUSH=1 has the most impact on performance, but will ensure that each entry is written to disk before the application continues to the next statement.

This option is only used when the TRACE CLI/ODBC option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343

TRACELOCKS CLI/ODBC Configuration Keyword

Keyword Description:
Only trace lock timeouts in the CLI/ODBC trace.

db2cli.ini Keyword Syntax:
TRACELOCKS = **0** | **1**

Default Setting:
Trace information is not limited to only lock timeouts.

Only Applicable when:
the TRACE option is turned on.

Usage Notes:

When TRACELOCKS is set to 1, lock timeouts will be recorded in the trace file.

This option is only used when the CLI/ODBC TRACE option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343

TRACEPATHNAME CLI/ODBC Configuration Keyword

Keyword Description:
Subdirectory used to store individual DB2 CLI/ODBC trace files.

db2cli.ini Keyword Syntax:
TRACEPATHNAME = < **Full subdirectory name** >

Default Setting:
None

Only Applicable when:
the TRACE option is turned on.

Not Applicable when:
the TRACEFILENAME option is turned on.

Usage Notes:

Each thread or process that uses the same DLL or shared library will have a separate DB2 CLI/ODBC trace file created in the specified directory.

No trace will occur, and no error message will be returned, if the subdirectory given is invalid or if it cannot be written to.

This option is only used when the TRACE option is turned on. This will be done automatically when you set this option in the CLI/ODBC Configuration utility.

It will be ignored if the DB2 CLI/ODBC option TRACEFILENAME is used.

DB2 CLI trace should only be used for debugging purposes. It will slow down the execution of the CLI/ODBC driver, and the trace information can grow quite large if it is left on for extended periods of time.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “db2cli.ini Initialization File” on page 293
- “CLI/ODBC/JDBC Trace Facility” on page 223

Related reference:

- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “TRACE CLI/ODBC Configuration Keyword” on page 339
- “TRACEFILENAME CLI/ODBC Configuration Keyword” on page 341
- “TRACEFLUSH CLI/ODBC Configuration Keyword” on page 342

TRACEPIDLIST CLI/ODBC Configuration Keyword

Keyword Description:
Restrict the process IDs for which the CLI/ODBC trace will be enabled.

db2cli.ini Keyword Syntax:
TRACEPIDLIST = **<no value specified>** | **<comma-delimited list of process IDs>**

Default Setting:
Any process ID will be traced when the CLI/ODBC trace is run.

Usage Notes:

If no value is specified for this keyword, all process IDs will be traced. Otherwise, specify a comma-delimited list of process IDs which you want to be traced when the CLI/ODBC trace runs.

For this keyword to be most effective, set the TRACEREFRESHINTERVAL keyword to some value before initializing your application.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword" on page 346
- "TRACEPIDTID CLI/ODBC Configuration Keyword" on page 345

TRACEPIDTID CLI/ODBC Configuration Keyword

Keyword Description:
Capture the process ID and thread ID for each item being traced.

db2cli.ini Keyword Syntax:
TRACEPIDTID = **0** | **1**

Default Setting:
The process ID and thread ID for the trace entries are not captured.

Only Applicable when:
the TRACE option is turned on.

Usage Notes:

When TRACEPIDTID is set to 1, the process ID and thread ID for each captured item will be recorded in the trace file. This effect is helpful when the TRACEFILENAME keyword is enabled and multiple applications are executing. This is because TRACEFILENAME writes trace information for all executing applications to a single file. Enabling TRACEPIDTID differentiates the recorded information by process and thread.

This option is only used when the CLI/ODBC TRACE option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343
- "TRACEPIDLIST CLI/ODBC Configuration Keyword" on page 344

TRACEREFRESHINTERVAL CLI/ODBC Configuration Keyword

Keyword Description:
Set the interval (in seconds) at which the TRACE and TRACEPIDLIST keywords are read from the Common section of the db2cli.ini file.

db2cli.ini Keyword Syntax:
TRACEREFRESHINTERVAL = **0** | **positive integer**

Default Setting:
The TRACE and TRACEPIDLIST keywords will only be read from the db2cli.ini file when the application is initialized.

Usage Notes:

Setting this keyword before an application is initialized allows you to turn off dynamically the CLI/ODBC trace within n seconds.

Note: Setting TRACEREFRESHINTERVAL while the application is running will no effect. For this keyword to take effect, it must be set before the application is initialized.

Only the TRACE and TRACEPIDLIST keywords will be refreshed from the db2cli.ini file if this keyword is set. No other CLI/ODBC configuration keywords will be re-read.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- "db2cli.ini Initialization File" on page 293
- "CLI/ODBC/JDBC Trace Facility" on page 223

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEPIDLIST CLI/ODBC Configuration Keyword" on page 344

TRACESTMTONLY CLI/ODBC Configuration Keyword

Keyword Description:
Only trace dynamic SQL statements in the CLI/ODBC trace.

db2cli.ini Keyword Syntax:
TRACESTMTONLY = 0 | **1**

Default Setting:
Trace information is not limited to only dynamic SQL statements.

Only Applicable when:
the TRACE option is turned on.

Usage Notes:

When TRACESTMTONLY is set to 1, only dynamic SQL statements will be recorded in the trace file.

This option is only used when the CLI/ODBC TRACE option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “db2cli.ini Initialization File” on page 293
- “CLI/ODBC/JDBC Trace Facility” on page 223

Related reference:

- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “TRACE CLI/ODBC Configuration Keyword” on page 339
- “TRACEFILENAME CLI/ODBC Configuration Keyword” on page 341
- “TRACEFLUSH CLI/ODBC Configuration Keyword” on page 342
- “TRACEPATHNAME CLI/ODBC Configuration Keyword” on page 343

TRACETIME CLI/ODBC Configuration Keyword

Keyword Description:
Capture elapsed time counters in the trace file.

db2cli.ini Keyword Syntax:
TRACETIME = **1** | **0**

Default Setting:
Elapsed time counters are included in the trace file.

Only Applicable when:
the TRACE option is turned on.

Usage Notes:

When TRACETIME is set to 1, elapsed time counters will be captured in the trace file. For example:

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
  ---> Time elapsed - +6.785751E+000 seconds ( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <--- SQL_SUCCESS Time elapsed - +2.527400E-002 seconds
```

Turn TRACETIME off, by setting it to 0, to improve performance or to generate smaller trace files. For example:

```
SQLPrepare( hStmt=1:1, pszSqlStr="SELECT * FROM ORG", cbSqlStr=-3 )
( StmtOut="SELECT * FROM ORG" )
SQLPrepare( )
  <--- SQL_SUCCESS
```

This option is only used when the CLI/ODBC TRACE option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:

- “db2cli.ini Initialization File” on page 293
- “CLI/ODBC/JDBC Trace Facility” on page 223

Related reference:
- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “TRACE CLI/ODBC Configuration Keyword” on page 339
- “TRACEFILENAME CLI/ODBC Configuration Keyword” on page 341
- “TRACEFLUSH CLI/ODBC Configuration Keyword” on page 342
- “TRACEPATHNAME CLI/ODBC Configuration Keyword” on page 343
- “TRACETIMESTAMP CLI/ODBC Configuration Keyword” on page 349

TRACETIMESTAMP CLI/ODBC Configuration Keyword

Keyword Description:
Capture different types of timestamp information in the CLI/ODBC trace.

db2cli.ini Keyword Syntax:
TRACETIMESTAMP = **0** | **1** | **2** | **3**

Default Setting:
No timestamp information is written to the trace file.

Only Applicable when:
the TRACE option is turned on.

Usage Notes:

The following settings indicate what type of timestamp information is captured in the trace file:
- 0 = no timestamp information
- 1 = ISO timestamp and processor ticks
- 2 = processor ticks
- 3 = ISO timestamp

This option is only used when the CLI/ODBC TRACE option is turned on.

(This option is contained in the Common section of the initialization file and therefore applies to all connections to DB2.)

Related concepts:
- “db2cli.ini Initialization File” on page 293
- “CLI/ODBC/JDBC Trace Facility” on page 223

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "TRACE CLI/ODBC Configuration Keyword" on page 339
- "TRACEFILENAME CLI/ODBC Configuration Keyword" on page 341
- "TRACEFLUSH CLI/ODBC Configuration Keyword" on page 342
- "TRACEPATHNAME CLI/ODBC Configuration Keyword" on page 343
- "TRACETIME CLI/ODBC Configuration Keyword" on page 348

TXNISOLATION CLI/ODBC Configuration Keyword

Keyword Description:
Set the default isolation level.

db2cli.ini Keyword Syntax:
TXNISOLATION = **1** | **2** | **4** | **8** | **32**

Default Setting:
Read Committed (Cursor Stability)

Only Applicable when:
the default isolation level is used. This keyword will have no effect if the application has specifically set the isolation level.

Equivalent Statement Attribute:
SQL_ATTR_TXN_ISOLATION

Usage Notes:

Sets the isolation level to:

- 1 = SQL_TXN_READ_UNCOMMITTED - Read Uncommitted (Uncommitted read)
- 2 = SQL_TXN_READ_COMMITTED (default) - Read Committed (Cursor stability)
- 4 = SQL_TXN_REPEATABLE_READ - Repeatable Read (Read Stability)
- 8 = SQL_TXN_SERIALIZABLE - Serializable (Repeatable read)
- 32 = SQL_TXN_NOCOMMIT - (No Commit, DB2 Universal Database for AS/400 only; this is similar to autocommit)

The words in parentheses are IBM's terminology for the equivalent SQL92 isolation levels. Note that *no commit* is not an SQL92 isolation level and is supported only on DB2 Universal Database for AS/400.

This keyword is only applicable if the default isolation level is used. If the application specifically sets the isolation level for a connection or statement handle, then this keyword will have no effect on that handle.

Related concepts:

- "Isolation levels" in the *SQL Reference, Volume 1*
- "db2cli.ini Initialization File" on page 293

Related reference:

- "Statement Attributes (CLI) List" in the *CLI Guide and Reference, Volume 2*
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

UID CLI/ODBC Configuration Keyword

Keyword Description:
Define default user ID.

db2cli.ini Keyword Syntax:
UID = *userid*

Default Setting:
None

Usage Notes:

The specified *userid* value is used if a userid is not provided by the application at connect time.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "PWD CLI/ODBC Configuration Keyword" on page 328

UNDERSCORE CLI/ODBC Configuration Keyword

Note: This keyword is not supported in DB2 Version 8, but is available for backward compatibility only. Refer to the documentation for previous versions of DB2 for information on this keyword at: http://www.ibm.com/software/data/db2/library.

Related concepts:

- "db2cli.ini Initialization File" on page 293

Related reference:

- "CLI/ODBC Configuration Keywords Listing by Category" on page 296

UNICODESERVER CLI/ODBC Configuration Keyword

Keyword Description:
Indicate the data source is a Unicode server. Enabling this keyword is equivalent to setting ConnectCodepage=1208.

db2cli.ini Keyword Syntax:
UNICODESERVER = **0** | **1**

Default Setting:
0

Usage Notes:

This keyword is equivalent to ConnectCodepage=1208, and is added only for convenience. Set this keyword to avoid extra connect overhead when connecting to DB2 for OS/390 Version 7 or z/OS. There is no need to set this keyword for DB2 for Windows or DB2 for Unix databases, because there is no extra processing required.

Related concepts:
- "db2cli.ini Initialization File" on page 293
- "Unicode CLI Applications" on page 167

Related reference:
- "CLI/ODBC Configuration Keywords Listing by Category" on page 296
- "CONNECTCODEPAGE CLI/ODBC Configuration Keyword" on page 304
- "DISABLEUNICODE CLI/ODBC Configuration Keyword" on page 318

USEOLDSTPCALL CLI/ODBC Configuration Keyword

Keyword Description:
Controls how cataloged procedures are invoked.

db2cli.ini Keyword Syntax:
USEOLDSTPCALL = **0** | **1**

Default Setting:
Invokes procedures using the new CALL method where GRANT EXECUTE must be granted on the procedure.

Usage Notes:

Prior to Version 8, the invoker of a procedure had to have EXECUTE privilege on any package invoked from the procedure. Now, the invoker must have

EXECUTE privilege on the procedure and only the definer of the procedure has to have EXECUTE privilege on any required packages.

This keyword controls which method is used to invoke the procedure. Setting USEOLDSTPCALL on causes the procedure to be invoked using the deprecated sqleproc() API when the precompiler fails to resolve a procedure on a CALL statement. Turning this keyword off will invoke procedures where GRANT EXECUTE must be granted on the procedure.

Related concepts:

- “db2cli.ini Initialization File” on page 293

Related tasks:

- “Calling Stored Procedures in CLI Applications” on page 143

Related reference:

- “CLI/ODBC Configuration Keywords Listing by Category” on page 296

WARNINGLIST CLI/ODBC Configuration Keyword

Keyword Description:
Specify which errors to downgrade to warnings.

db2cli.ini Keyword Syntax:
WARNINGLIST = ″ **’xxxxx’**, **’yyyyy’**, ...″

Default Setting:
Do not downgrade any SQLSTATEs.

Usage Notes:

Any number of SQLSTATEs returned as errors can be downgraded to warnings. Each must be delimited with single quotes, separated by commas, and in uppercase. The entire string must also be enclosed in double quotes. For example:

```
WARNINGLIST=" '01S02', 'HY090' "
```

Related concepts:

- “db2cli.ini Initialization File” on page 293

Related reference:

- “CLI/ODBC Configuration Keywords Listing by Category” on page 296
- “IGNOREWARNLIST CLI/ODBC Configuration Keyword” on page 321

Part 5. Data Conversion

Chapter 26. Data Conversion

This chapter describes the data conversions supported in CLI, along with the details of SQL to C and C to SQL data type conversions.

Data Conversions Supported in CLI

The table below shows which data type conversions are supported by DB2 CLI.

The first column contains the data type of the SQL data type. The remaining columns represent the C data types. If the C data type columns contain:

D The conversion is supported and this is the default conversion for the SQL data type.

X all IBM DBMSs support the conversion.

blank no IBM DBMS supports the conversion.

As an example, the table indicates that a CHAR (or a C character string can be converted into a SQL_C_LONG (a signed long). In contrast, a LONGVARCHAR cannot be converted to a SQL_C_LONG.

Refer to the tables of data type attributes (precision, scale, length, and display) for more information on the data type formats.

Table 20. Supported Data Conversions

SQL Data Type	SQL_C_CHAR	SQL_C_BINARY	SQL_C_WCHAR	SQL_C_DBCHAR	SQL_C_SHORT	SQL_C_USHORT	SQL_C_LONG	SQL_C_ULONG	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_TINYINT	SQL_C_UTINYINT	SQL_C_BIT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_NUMERIC	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_DECIMAL_IBM	SQL_C_DATALINK	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
SQL_CHAR	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
SQL_VARCHAR	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_LONG VARCHAR	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_BINARY	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
SQL_VARBINARY	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_LONG VARBINARY	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			
SQL_GRAPHIC	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
SQL_VARGRAPHIC	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
SQL_LONG VARGRAPHIC	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				
SQL_CLOB	D	X	X	X																		X		
SQL_BLOB	X	D	X	X																			X	
SQL_DBCLOB	X	X	X	D																				X
SQL_CLOB_LOCATOR																						D		
SQL_BLOB_LOCATOR																							D	
SQL_DBCLOB_LOCATOR																								D
SQL_NUMERIC	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X				

Table 20. Supported Data Conversions (continued)

SQL Data Type	SQL_C_CHAR	SQL_C_BINARY	SQL_C_WCHAR	SQL_C_DBCHAR	SQL_C_SHORT	SQL_C_USHORT	SQL_C_LONG	SQL_C_ULONG	SQL_C_SBIGINT	SQL_C_UBIGINT	SQL_C_TINYINT	SQL_C_UTINYINT	SQL_C_BIT	SQL_C_FLOAT	SQL_C_DOUBLE	SQL_C_NUMERIC	SQL_C_TYPE_DATE	SQL_C_TYPE_TIME	SQL_C_TYPE_TIMESTAMP	SQL_C_DECIMAL_IBM	SQL_C_DATALINK	SQL_C_CLOB_LOCATOR	SQL_C_BLOB_LOCATOR	SQL_C_DBCLOB_LOCATOR
SQL_DECIMAL	D	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X				
SQL_INTEGER	X	X	X	X	X	X	D	X	X	X	X	X	X	X	X	X								
SQL_ SMALLINT	X	X	X	X	D	X	X	X	X	X	X	X	X	X	X	X								
SQL_FLOAT	X	X	X	X	X	X	X	X	X	X	X	X	X	X	D	X								
SQL_DOUBLE	X	X	X	X	X	X	X	X	X	X	X	X	X	X	D	X								
SQL_REAL	X	X	X	X	X	X	X	X	X	X	X	X	X	D	X	X								
SQL_BIGINT	X	X	X	X	X	X	X	X	D	X	X	X	X	X	X	X								
SQL_TINYINT	X	X	X	X	X	X	X	X	X	X	D	X	X	X	X	X								
SQL_BIT	X	X	X	X	X	X	X	X	X	X	X	X	D	X	X	X								
SQL_ TYPE_DATE	X		X	X													D		X					
SQL_ TYPE_TIME	X		X	X														D	X					
SQL_ TYPE_ TIMESTAMP	X		X	X													X	X	D					
SQL_ DATALINK	D	X	X	X																				

Related concepts:

- "Data Types and Data Conversion in CLI Applications" on page 49

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "SQL to C Data Conversion in CLI" on page 360
- "C to SQL Data Conversion in CLI" on page 368

SQL to C Data Conversion in CLI

For a given SQL data type:

- the first column of the table lists the legal input values of the *fCType* argument in `SQLBindCol()` and `SQLGetData()`.
- the second column lists the outcomes of a test, often using the *cbValueMax* argument specified in `SQLBindCol()` or `SQLGetData()`, which the driver performs to determine if it can convert the data.
- the third and fourth columns list the values (for each outcome) of the *rgbValue* and *pcbValue* arguments specified in the `SQLBindCol()` or `SQLGetData()` after the driver has attempted to convert the data.
- the last column lists the SQLSTATE returned for each outcome by `SQLFetch()`, `SQLExtendedFetch()`, `SQLGetData()` or `SQLGetSubString()`.

The tables list the conversions defined by ODBC to be valid for a given SQL data type.

If the *fCType* argument in `SQLBindCol()` or `SQLGetData()` contains a value not shown in the table for a given SQL data type, `SQLFetch()`, or `SQLGetData()` returns the SQLSTATE 07006 (Restricted data type attribute violation).

If the *fCType* argument contains a value shown in the table but which specifies a conversion not supported by the driver, `SQLFetch()`, or `SQLGetData()` returns SQLSTATE HYC00 (Driver not capable).

Though it is not shown in the tables, the *pcbValue* argument contains SQL_NULL_DATA when the SQL data value is NULL. For an explanation of the use of *pcbValue* when multiple calls are made to retrieve data, see `SQLGetData()`.

When SQL data is converted to character C data, the character count returned in *pcbValue* does not include the null termination byte. If *rgbValue* is a null pointer, `SQLBindCol()` or `SQLGetData()` returns SQLSTATE HY009 (Invalid argument value).

In the following tables:

Length of data
: the total length of the data after it has been converted to the specified C data type (excluding the null termination byte if the data was converted to a string). This is true even if data is truncated before it is returned to the application.

Significant digits
: the minus sign (if needed) and the digits to the left of the decimal point.

Display size
: the total number of bytes needed to display data in the character format.

Converting Character SQL Data to C Data:

The character SQL data types are:
- SQL_CHAR
- SQL_VARCHAR
- SQL_LONGVARCHAR
- SQL_CLOB

Table 21. Converting Character SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Length of data < cbValueMax	Data	Length of data	00000
	Length of data >= cbValueMax	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	00000
	Length of data > cbValueMax	Truncated data	Length of data	01004
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_FLOAT SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [c]	Data converted without truncation [a]	Data	Size of the C data type	00000
	Data converted with truncation, but without loss of significant digits [a]	Data	Size of the C data type	01004
	Conversion of data would result in loss of significant digits[a]	Untouched	Size of the C data type	22003
	Data is not a number [a]	Untouched	Size of the C data type	22005
SQL_C_DATE	Data value is a valid date [a]	Data	6 [b]	00000
	Data value is not a valid date [a]	Untouched	6 [b]	22007
SQL_C_TIME	Data value is a valid time [a]	Data	6 [b]	00000
	Data value is not a valid time [a]	Untouched	6 [b]	22007
SQL_C_TIMESTAMP	Data value is a valid timestamp [a]	Data	16 [b]	00000
	Data value is not a valid timestamp [a]	Untouched	16 [b]	22007

Note:

[a] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[b] This is the size of the corresponding C data type.

[c] SQL_C_NUMERIC is only supported on Windows platforms.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Converting Graphic SQL Data to C Data:

The graphic SQL data types are:
SQL_GRAPHIC
SQL_VARGRAPHIC
SQL_LONGVARGRAPHIC
SQL_DBCLOB

Table 22. Converting Graphic SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Number of double byte characters * 2 <= cbValueMax	Data	Length of data(octects)	00000
	Number of double byte characters * 2 > cbValueMax	Truncated data, to the nearest even byte that is less than *cbValueMax*.	Length of data(octects)	01004
SQL_C_DBCHAR	Number of double byte characters * 2 < cbValueMax	Data	Length of data(octects)	00000
	Number of double byte characters * 2 >= cbValueMax	Truncated data, to the nearest even byte that is less than *cbValueMax*.	Length of data(octects)	01004

Note: SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

When converting to floating point values, SQLSTATE 22003 will not be returned if non-significant digits of the resulting value are lost.

Converting Numeric SQL Data to C Data:

The numeric SQL data types are:
SQL_DECIMAL
SQL_NUMERIC
SQL_SMALLINT
SQL_INTEGER

SQL_BIGINT
SQL_REAL
SQL_FLOAT
SQL_DOUBLE

Table 23. Converting Numeric SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Display size < cbValueMax	Data	Length of data	00000
	Number of significant digits < cbValueMax	Truncated data	Length of data	01004
	Number of significant digits >= cbValueMax	Untouched	Length of data	22003
SQL_C_SHORT SQL_C_LONG SQL_C_FLOAT SQL_C_DOUBLE SQL_C_TINYINT SQL_C_BIT SQL_C_UBIGINT SQL_C_SBIGINT SQL_C_NUMERIC [b]	Data converted without truncation [a]	Data	Size of the C data type	00000
	Data converted with truncation, but without loss of significant digits [a]	Truncated data	Size of the C data type	01004
	Conversion of data would result in loss of significant digits [a]	Untouched	Size of the C data type	22003

Note:

[a] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[b] SQL_C_NUMERIC is only supported on Windows platforms.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Converting Binary SQL Data to C Data:

The binary SQL data types are:
SQL_BINARY
SQL_VARBINARY
SQL_LONGVARBINARY
SQL_BLOB

Table 24. Converting Binary SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	(Length of data) < cbValueMax	Data	Length of data	N/A
	(Length of data) >= cbValueMax	Truncated data	Length of data	01004
SQL_C_BINARY	Length of data <= cbValueMax	Data	Length of data	N/A
	Length of data > cbValueMax	Truncated data	Length of data	01004

Converting Date SQL Data to C Data:

The date SQL data type is:
  SQL_DATE

Table 25. Converting Date SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 11	Data	10	00000
	cbValueMax < 11	Untouched	10	22003
SQL_C_DATE	None [a]	Data	6 [b]	00000
SQL_C_TIMESTAMP	None [a]	Data [c]	16 [b]	00000

Note:

[a] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[b] This is the size of the corresponding C data type.

[c] The time fields of the TIMESTAMP_STRUCT structure are set to zero.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

When the date SQL data type is converted to the character C data type, the resulting string is in the "yyyy-mm-dd” format.

Converting Time SQL Data to C Data:

The time SQL data type is:
  SQL_TIME

Table 26. Converting Time SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	cbValueMax >= 9	Data	8	00000
	cbValueMax < 9	Untouched	8	22003
SQL_C_TIME	None [a]	Data	6 [b]	00000
SQL_C_TIMESTAMP	None [a]	Data [c]	16 [b]	00000

Note:

[a] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[b] This is the size of the corresponding C data type.

[c] The date fields of the TIMESTAMP_STRUCT structure are set to the current system date of the machine that the application is running, and the time fraction is set to zero.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

When the time SQL data type is converted to the character C data type, the resulting string is in the "hh:mm:ss” format.

Converting Timestamp SQL Data to C Data:

The timestamp SQL data type is:
SQL_TIMESTAMP

Table 27. Converting Timestamp SQL Data to C Data

fCType	Test	rgbValue	pcbValue	SQLSTATE
SQL_C_CHAR	Display size < cbValueMax	Data	Length of data	00000
	19 <= cbValueMax <= Display size	Truncated Data [b]	Length of data	01004
	cbValueMax < 19	Untouched	Length of data	22003
SQL_C_DATE	None [a]	Truncated data [c]	6 [e]	01004
SQL_C_TIME	None [a]	Truncated data [d]	6 [e]	01004
SQL_C_TIMESTAMP	None [a]	Data	16 [e]	00000

Note:

[a] The value of *cbValueMax* is ignored for this conversion. The driver assumes that the size of *rgbValue* is the size of the C data type.

[b] The fractional seconds of the timestamp are truncated.

[c] The time portion of the timestamp is deleted.

[d] The date portion of the timestamp is deleted.

[e] This is the size of the corresponding C data type.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

When the timestamp SQL data type is converted to the character C data type, the resulting string is in the "yyyy-mm-dd hh:mm:ss.ffffff” format (regardless of the precision of the timestamp SQL data type).

SQL to C Data Conversion Examples:

Table 28. SQL to C Data Conversion Examples

SQL Data Type	SQL Data Value	C Data Type	cbValue Max	rgbValue	SQL STATE
SQL_CHAR	abcdef	SQL_C_CHAR	7	abcdef\0 [a]	00000
SQL_CHAR	abcdef	SQL_C_CHAR	6	abcde\0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	8	1234.56\0 [a]	00000
SQL_DECIMAL	1234.56	SQL_C_CHAR	5	1234\0 [a]	01004
SQL_DECIMAL	1234.56	SQL_C_CHAR	4	---	22003
SQL_DECIMAL	1234.56	SQL_C_FLOAT	ignored	1234.56	00000
SQL_DECIMAL	1234.56	SQL_C_SHORT	ignored	1234	01004

Table 28. SQL to C Data Conversion Examples (continued)

SQL Data Type	SQL Data Value	C Data Type	cbValue Max	rgbValue	SQL STATE
SQL_DATE	1992-12-31	SQL_C_CHAR	11	1992-12-31\0 [a]	00000
SQL_DATE	1992-12-31	SQL_C_CHAR	10	---	22003
SQL_DATE	1992-12-31	SQL_C_TIMESTAMP	ignored	1992,12,31, 0,0,0,0 [b]	00000
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	23	1992-12-31 23:45:55.12\0 [a]	00000
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	22	1992-12-31 23:45:55.1\0 [a]	01004
SQL_TIMESTAMP	1992-12-31 23:45:55.12	SQL_C_CHAR	18	---	22003

Note:

a ″\0″ represents a null termination character.

b The numbers in this list are the numbers stored in the fields of the TIMESTAMP_STRUCT structure.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "Data Conversions Supported in CLI" on page 357
- "C to SQL Data Conversion in CLI" on page 368

C to SQL Data Conversion in CLI

For a given C data type:

- the first column of the table lists the legal input values of the *fSqlType* argument in `SQLBindParameter()` or `SQLSetParam()`.
- the second column lists the outcomes of a test, often using the length of the parameter data as specified in the *pcbValue* argument in `SQLBindParameter()` or `SQLSetParam()`, which the driver performs to determine if it can convert the data.

- the third column lists the SQLSTATE returned for each outcome by `SQLExecDirect()` or `SQLExecute()`.

The tables list the conversions defined by ODBC to be valid for a given SQL data type.

If the *fSqlType* argument in `SQLBindParameter()` or `SQLSetParam()` contains a value not shown in the table for a given C data type, SQLSTATE 07006 is returned (Restricted data type attribute violation).

If the *fSqlType* argument contains a value shown in the table but which specifies a conversion not supported by the driver, `SQLBindParameter()` or `SQLSetParam()` returns SQLSTATE HYC00 (Driver not capable).

If the *rgbValue* and *pcbValue* arguments specified in `SQLBindParameter()` or `SQLSetParam()` are both null pointers, that function returns SQLSTATE HY009 (Invalid argument value).

Length of data
: the total length of the data after it has been converted to the specified SQL data type (excluding the null termination byte if the data was converted to a string). This is true even if data is truncated before it is sent to the data source.

Column length
: the maximum number of bytes returned to the application when data is transferred to its default C data type. For character data, the length does not include the null termination byte.

Display size
: the maximum number of bytes needed to display data in character form.

Significant digits
: the minus sign (if needed) and the digits to the left of the decimal point.

Converting Character C Data to SQL Data:

The character C data type is:
SQL_C_CHAR

Table 29. Converting Character C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data <= Column length	N/A
	Length of data > Column length	22001
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_REAL SQL_FLOAT SQL_DOUBLE	Data converted without truncation	N/A
	Data converted with truncation, but without loss of significant digits	22001
	Conversion of data would result in loss of significant digits	22003
	Data value is not a numeric value	22005
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	(Length of data) < Column length	N/A
	(Length of data) >= Column length	22001
	Data value is not a hexadecimal value	22005
SQL_DATE	Data value is a valid date	N/A
	Data value is not a valid date	22007
SQL_TIME	Data value is a valid time	N/A
	Data value is not a valid time	22007
SQL_TIMESTAMP	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007
SQL_GRAPHIC SQL_VARGRAPHIC SQL_LONGVARGRAPHIC SQL_DBCLOB	Length of data / 2 <= Column length	N/A
	Length of data / 2 < Column length	22001

Note: SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Converting Numeric C Data to SQL Data:

The numeric C data types are:
- SQL_C_SHORT
- SQL_C_LONG
- SQL_C_FLOAT
- SQL_C_DOUBLE
- SQL_C_TINYINT
- SQL_C_BIT

Table 30. Converting Numeric C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_DECIMAL SQL_NUMERIC SQL_SMALLINT SQL_INTEGER SQL_REAL SQL_FLOAT SQL_DOUBLE	Data converted without truncation	N/A
	Data converted with truncation, but without loss of significant digits	22001
	Conversion of data would result in loss of significant digits	22003
SQL_CHAR SQL_VARCHAR	Data converted without truncation.	N/A
	Conversion of data would result in loss of significant digits.	22003

Note: SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

When converting to floating point values, SQLSTATE 22003 will not be returned if non-significant digits of the resulting value are lost.

Converting Binary C Data to SQL Data:

The binary C data type is:
  SQL_C_BINARY

Table 31. Converting Binary C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data <= Column length	N/A
	Length of data > Column length	22001
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	Length of data <= Column length	N/A
	Length of data > Column length	22001

Converting DBCHAR C Data to SQL Data:

The Double Byte C data type is:
  SQL_C_DBCHAR

Table 32. Converting DBCHAR C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR SQL_LONGVARCHAR SQL_CLOB	Length of data <= Column length x 2	N/A
	Length of data > Column length x 2	22001
SQL_BINARY SQL_VARBINARY SQL_LONGVARBINARY SQL_BLOB	Length of data <= Column length x 2	N/A
	Length of data > Column length x 2	22001

Converting Date C Data to SQL Data:

The date C data type is:
SQL_C_DATE

Table 33. Converting Date C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= 10	N/A
	Column length < 10	22003
SQL_DATE	Data value is a valid date	N/A
	Data value is not a valid date	22007
SQL_TIMESTAMP [a]	Data value is a valid date	N/A
	Data value is not a valid date	22007

Note: SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Note: **a**, the time component of TIMESTAMP is set to zero.

Converting Time C Data to SQL Data:

The time C data type is:
SQL_C_TIME

Table 34. Converting Time C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= 8	N/A
	Column length < 8	22003
SQL_TIME	Data value is a valid time	N/A
	Data value is not a valid time	22007
SQL_TIMESTAMP [a]	Data value is a valid time	N/A
	Data value is not a valid time	22007

Note: SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Note: **a** The date component of TIMESTAMP is set to the system date of the machine at which the application is running.

Converting Timestamp C Data to SQL Data:

The timestamp C data type is:
SQL_C_TIMESTAMP

Table 35. Converting Timestamp C Data to SQL Data

fSQLType	Test	SQLSTATE
SQL_CHAR SQL_VARCHAR	Column length >= Display size	N/A
	19 <= Column length < Display size [a]	22001
	Column length < 19	22003
SQL_DATE	Time fields are zero	N/A
	Time fields are non-zero	22008
Data value does not contain a valid date	22007	
SQL_TIME	Fractional seconds fields are zero	N/A
	Fractional seconds fields are non-zero	22008
Data value does not contain a valid time	22007	
SQL_TIMESTAMP	Data value is a valid timestamp	N/A
	Data value is not a valid timestamp	22007

Note:
a The fractional seconds of the timestamp are truncated.

SQLSTATE **00**000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

C to SQL Data Conversion Examples:

Table 36. C to SQL Data Conversion Examples

C Data Type	C Data Value	SQL Data Type	Column length	SQL Data Value	SQL STATE
SQL_C_CHAR	abcdef\0	SQL_CHAR	6	abcdef	N/A
SQL_C_CHAR	abcdef\0	SQL_CHAR	5	abcde	22001
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	6	1234.56	N/A
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	5	1234.5	22001
SQL_C_CHAR	1234.56\0	SQL_DECIMAL	3	---	22003
SQL_C_CHAR	4.46.32	SQL_TIME	6	4.46.32	N/A
SQL_C_CHAR	4-46-32	SQL_TIME	6	not applicable	22007
SQL_C_DOUBLE	123.45	SQL_CHAR	22	1.23450000 000000e+02	N/A
SQL_C_FLOAT	1234.56	SQL_FLOAT	not applicable	1234.56	N/A
SQL_C_FLOAT	1234.56	SQL_INTEGER	not applicable	1234	22001
SQL_C_ TIMESTAMP	1992-12-31 23:45:55. 123456	SQL_DATE	6	1992-12-31	01004

Note: SQLSTATE 00000 is not returned by `SQLError()`, rather it is indicated when the function returns SQL_SUCCESS.

Related reference:

- "SQL Symbolic and Default Data Types for CLI Applications" on page 51
- "C Data Types for CLI Applications" on page 54
- "Data Conversions Supported in CLI" on page 357
- "SQL to C Data Conversion in CLI" on page 360

Part 6. Appendixes

Appendix A. DB2 Universal Database technical information

Overview of DB2 Universal Database technical information

DB2 Universal Database technical information can be obtained in the following formats:

- Books (PDF and hard-copy formats)
- A topic tree (HTML format)
- Help for DB2 tools (HTML format)
- Sample programs (HTML format)
- Command line help
- Tutorials

This section is an overview of the technical information that is provided and how you can access it.

Categories of DB2 technical information

The DB2 technical information is categorized by the following headings:

- Core DB2 information
- Administration information
- Application development information
- Business intelligence information
- DB2 Connect information
- Getting started information
- Tutorial information
- Optional component information
- Release notes

The following tables describe, for each book in the DB2 library, the information needed to order the hard copy, print or view the PDF, or locate the HTML directory for that book. A full description of each of the books in the DB2 library is available from the IBM Publications Center at www.ibm.com/shop/publications/order

The installation directory for the HTML documentation CD differs for each category of information:

```
htmlcdpath/doc/htmlcd/%L/category
```

where:

- *htmlcdpath* is the directory where the HTML CD is installed.
- *%L* is the language identifier. For example, en_US.
- *category* is the category identifier. For example, core for the core DB2 information.

In the PDF file name column in the following tables, the character in the sixth position of the file name indicates the language version of a book. For example, the file name `db2d1e80` identifies the English version of the *Administration Guide: Planning* and the file name `db2d1g80` identifies the German version of the same book. The following letters are used in the sixth position of the file name to indicate the language version:

Language	**Identifier**
Arabic	w
Brazilian Portuguese	b
Bulgarian	u
Croatian	9
Czech	x
Danish	d
Dutch	q
English	e
Finnish	y
French	f
German	g
Greek	a
Hungarian	h
Italian	i
Japanese	j
Korean	k
Norwegian	n
Polish	p
Portuguese	v
Romanian	8
Russian	r
Simp. Chinese	c
Slovakian	7
Slovenian	l
Spanish	z
Swedish	s
Trad. Chinese	t
Turkish	m

No form number indicates that the book is only available online and does not have a printed version.

Core DB2 information

The information in this category cover DB2 topics that are fundamental to all DB2 users. You will find the information in this category useful whether you are a programmer, a database administrator, or you work with DB2 Connect, DB2 Warehouse Manager, or other DB2 products.

The installation directory for this category is doc/htmlcd/*%L*/core.

Table 37. Core DB2 information

Name	Form Number	PDF File Name
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
IBM DB2 Universal Database Glossary	No form number	db2t0x80
IBM DB2 Universal Database Master Index	SC09-4839	db2w0x80
IBM DB2 Universal Database Message Reference, Volume 1	GC09-4840	db2m1x80
IBM DB2 Universal Database Message Reference, Volume 2	GC09-4841	db2m2x80
IBM DB2 Universal Database What's New	SC09-4848	db2q0x80

Administration information

The information in this category covers those topics required to effectively design, implement, and maintain DB2 databases, data warehouses, and federated systems.

The installation directory for this category is doc/htmlcd/*%L*/admin.

Table 38. Administration information

Name	Form number	PDF file name
IBM DB2 Universal Database Administration Guide: Planning	SC09-4822	db2d1x80
IBM DB2 Universal Database Administration Guide: Implementation	SC09-4820	db2d2x80
IBM DB2 Universal Database Administration Guide: Performance	SC09-4821	db2d3x80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80

Table 38. Administration information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
IBM DB2 Universal Database Data Recovery and High Availability Guide and Reference	SC09-4831	db2hax80
IBM DB2 Universal Database Data Warehouse Center Administration Guide	SC27-1123	db2ddx80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
IBM DB2 Universal Database Guide to GUI Tools for Administration and Development	SC09-4851	db2atx80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
IBM DB2 Installing and Administering a Satellite Environment	GC09-4823	db2dsx80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

Application development information

The information in this category is of special interest to application developers or programmers working with DB2. You will find information about supported languages and compilers, as well as the documentation required to access DB2 using the various supported programming interfaces, such as embedded SQL, ODBC, JDBC, SQLj, and CLI. If you view this information online in HTML you can also access a set of DB2 sample programs in HTML.

The installation directory for this category is doc/htmlcd/%*L*/ad.

Table 39. Application development information

Name	Form number	PDF file name
IBM DB2 Universal Database Application Development Guide: Building and Running Applications	SC09-4825	db2axx80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

Business intelligence information

The information in this category describes how to use components that enhance the data warehousing and analytical capabilities of DB2 Universal Database.

The installation directory for this category is doc/htmlcd/%*L*/wareh.

Table 40. Business intelligence information

Name	Form number	PDF file name
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
IBM DB2 Warehouse Manager Installation Guide	GC27-1122	db2idx80

DB2 Connect information

The information in this category describes how to access host or iSeries data using DB2 Connect Enterprise Edition or DB2 Connect Personal Edition.

The installation directory for this category is doc/htmlcd/*%L*/conn.

Table 41. DB2 Connect information

Name	Form number	PDF file name
APPC, CPI-C, and SNA Sense Codes	No form number	db2apx80
IBM Connectivity Supplement	No form number	db2h1x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Enterprise Edition	GC09-4833	db2c6x80
IBM DB2 Connect Quick Beginnings for DB2 Connect Personal Edition	GC09-4834	db2c1x80
IBM DB2 Connect User's Guide	SC09-4835	db2c0x80

Getting started information

The information in this category is useful when you are installing and configuring servers, clients, and other DB2 products.

The installation directory for this category is doc/htmlcd/*%L*/start.

Table 42. Getting started information

Name	Form number	PDF file name
IBM DB2 Universal Database Quick Beginnings for DB2 Clients	GC09-4832	db2itx80
IBM DB2 Universal Database Quick Beginnings for DB2 Servers	GC09-4836	db2isx80
IBM DB2 Universal Database Quick Beginnings for DB2 Personal Edition	GC09-4838	db2i1x80
IBM DB2 Universal Database Installation and Configuration Supplement	GC09-4837	db2iyx80
IBM DB2 Universal Database Quick Beginnings for DB2 Data Links Manager	GC09-4829	db2z6x80

Tutorial information

Tutorial information introduces DB2 features and teaches how to perform various tasks.

The installation directory for this category is doc/htmlcd/*%L*/tutr.

Table 43. Tutorial information

Name	Form number	PDF file name
Business Intelligence Tutorial: Introduction to the Data Warehouse	No form number	db2tux80
Business Intelligence Tutorial: Extended Lessons in Data Warehousing	No form number	db2tax80
Development Center Tutorial for Video Online using Microsoft Visual Basic	No form number	db2tdx80
Information Catalog Center Tutorial	No form number	db2aix80
Video Central for e-business Tutorial	No form number	db2twx80
Visual Explain Tutorial	No form number	db2tvx80

Optional component information

The information in this category describes how to work with optional DB2 components.

The installation directory for this category is doc/htmlcd/*%L*/opt.

Table 44. Optional component information

Name	Form number	PDF file name
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Universal Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80

Table 44. Optional component information (continued)

Name	Form number	PDF file name
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **Note:** HTML for this document is not installed from the HTML documentation CD.	SH12-6740	N/A

Release notes

The release notes provide additional information specific to your product's release and FixPak level. They also provides summaries of the documentation updates incorporated in each release and FixPak.

Table 45. Release notes

Name	Form number	PDF file name	HTML directory
DB2 Release Notes	See note.	See note.	doc/prodcd/*%L*/db2ir where *%L* is the language identifier.
DB2 Connect Release Notes	See note.	See note.	doc/prodcd/*%L*/db2cr where *%L* is the language identifier.
DB2 Installation Notes	Available on product CD-ROM only.	Available on product CD-ROM only.	

Note: The HTML version of the release notes is available from the Information Center and on the product CD-ROMs. To view the ASCII file:

- On UNIX-based platforms, see the Release.Notes file. This file is located in the DB2DIR/Readme/*%L* directory, where `%L` represents the locale name and `DB2DIR` represents:
  - /usr/opt/db2_08_01 on AIX
  - /opt/IBM/db2/V8.1 on all other UNIX operating systems
- On other platforms, see the RELEASE.TXT file. This file is located in the directory where the product is installed.

Related tasks:

- "Printing DB2 books from PDF files" on page 385

- “Ordering printed DB2 books” on page 386
- “Accessing online help” on page 386
- “Finding product information by accessing the DB2 Information Center from the administration tools” on page 390
- “Viewing technical documentation online directly from the DB2 HTML Documentation CD” on page 391

Printing DB2 books from PDF files

You can print DB2 books from the PDF files on the *DB2 PDF Documentation* CD. Using Adobe Acrobat Reader, you can print either the entire book or a specific range of pages.

Prerequisites:

Ensure that you have Adobe Acrobat Reader. It is available from the Adobe Web site at www.adobe.com

Procedure:

To print a DB2 book from a PDF file:

1. Insert the *DB2 PDF Documentation* CD. On UNIX operating systems, mount the DB2 PDF Documentation CD. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start Adobe Acrobat Reader.
3. Open the PDF file from one of the following locations:
   - On Windows operating systems:

     x:\doc\language directory, where *x* represents the CD-ROM drive letter and *language* represents the two-character territory code that represents your language (for example, EN for English).
   - On UNIX operating systems:

     /cdrom/doc/*%L* directory on the CD-ROM, where */cdrom* represents the mount point of the CD-ROM and *%L* represents the name of the desired locale.

Related tasks:

- “Ordering printed DB2 books” on page 386
- “Finding product information by accessing the DB2 Information Center from the administration tools” on page 390
- “Viewing technical documentation online directly from the DB2 HTML Documentation CD” on page 391

Related reference:

- "Overview of DB2 Universal Database technical information" on page 377

Ordering printed DB2 books

Procedure:

To order printed books:

- Contact your IBM authorized dealer or marketing representative. To find a local IBM representative, check the IBM Worldwide Directory of Contacts at www.ibm.com/shop/planetwide
- Phone `1-800-879-2755` in the United States or `1-800-IBM-4YOU` in Canada.
- Visit the IBM Publications Center at www.ibm.com/shop/publications/order

Related tasks:

- "Printing DB2 books from PDF files" on page 385
- "Finding topics by accessing the DB2 Information Center from a browser" on page 388
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 391

Related reference:

- "Overview of DB2 Universal Database technical information" on page 377

Accessing online help

The online help that comes with all DB2 components is available in three types:

- Window and notebook help
- Command line help
- SQL statement help

Window and notebook help explain the tasks that you can perform in a window or notebook and describe the controls. This help has two types:

- Help accessible from the **Help** button
- Infopops

The **Help** button gives you access to overview and prerequisite information. The infopops describe the controls in the window or notebook. Window and notebook help are available from DB2 centers and components that have user interfaces.

Command line help includes Command help and Message help. Command help explains the syntax of commands in the command line processor. Message help describes the cause of an error message and describes any action you should take in response to the error.

SQL statement help includes SQL help and SQLSTATE help. DB2 returns an SQLSTATE value for conditions that could be the result of an SQL statement. SQLSTATE help explains the syntax of SQL statements (SQL states and class codes).

Note: SQL help is not available for UNIX operating systems.

Procedure:

To access online help:

- For window and notebook help, click **Help** or click that control, then click **F1**. If the **Automatically display infopops** check box on the **General** page of the **Tool Settings** notebook is selected, you can also see the infopop for a particular control by holding the mouse cursor over the control.
- For command line help, open the command line processor and enter:
  - For Command help:
    ```
    ? command
    ```
    where *command* represents a keyword or the entire command.

  For example, `? catalog` displays help for all the CATALOG commands, while `? catalog database` displays help for the CATALOG DATABASE command.
- For Message help:
  ```
  ? XXXnnnnn
  ```
  where *XXXnnnnn* represents a valid message identifier.

  For example, `? SQL30081` displays help about the SQL30081 message.
- For SQL statement help, open the command line processor and enter:
  - For SQL help:
    ```
    ? sqlstate or ? class code
    ```
    where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

    For example, `? 08003` displays help for the 08003 SQL state, while `? 08` displays help for the 08 class code.
  - For SQLSTATE help:

```
help statement
```

where *statement* represents an SQL statement.

For example, help SELECT displays help about the SELECT statement.

Related tasks:
- "Finding topics by accessing the DB2 Information Center from a browser" on page 388
- "Viewing technical documentation online directly from the DB2 HTML Documentation CD" on page 391

Finding topics by accessing the DB2 Information Center from a browser

The DB2 Information Center accessed from a browser enables you to access the information you need to take full advantage of DB2 Universal Database and DB2 Connect. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, metadata, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser is composed of the following major elements:

Navigation tree
The navigation tree is located in the left frame of the browser window. The tree expands and collapses to show and hide topics, the glossary, and the master index in the DB2 Information Center.

Navigation toolbar
The navigation toolbar is located in the top right frame of the browser window. The navigation toolbar contains buttons that enable you to search the DB2 Information Center, hide the navigation tree, and find the currently displayed topic in the navigation tree.

Content frame
The content frame is located in the bottom right frame of the browser window. The content frame displays topics from the DB2 Information Center when you click on a link in the navigation tree, click on a search result, or follow a link from another topic or from the master index.

Prerequisites:

To access the DB2 Information Center from a browser, you must use one of the following browsers:
- Microsoft Explorer, version 5 or later
- Netscape Navigator, version 6.1 or later

Restrictions:

The DB2 Information Center contains only those sets of topics that you chose to install from the *DB2 HTML Documentation CD*. If your Web browser returns a `File not found` error when you try to follow a link to a topic, you must install one or more additional sets of topics *DB2 HTML Documentation CD*.

Procedure:

To find a topic by searching with keywords:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

To find a topic in the navigation tree:

1. In the navigation tree, click the book icon of the category of topics related to your area of interest. A list of subcategories displays underneath the icon.
2. Continue to click the book icons until you find the category containing the topics in which you are interested. Categories that link to topics display the category title as an underscored link when you move the cursor over the category title. The navigation tree identifies topics with a page icon.
3. Click the topic link. The topic displays in the content frame.

To find a topic or term in the master index:

1. In the navigation tree, click the “Index” category. The category expands to display a list of links arranged in alphabetical order in the navigation tree.
2. In the navigation tree, click the link corresponding to the first character of the term relating to the topic in which you are interested. A list of terms with that initial character displays in the content frame. Terms that have multiple index entries are identified by a book icon.
3. Click the book icon corresponding to the term in which you are interested. A list of subterms and topics displays below the term you clicked. Topics are identified by page icons with an underscored title.
4. Click on the title of the topic that meets your needs. The topic displays in the content frame.

Related concepts:
- "Accessibility" on page 397
- "DB2 Information Center for topics" on page 399

Related tasks:
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 390
- "Updating the HTML documentation installed on your machine" on page 392
- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 394
- "Searching the DB2 documentation" on page 395

Related reference:
- "Overview of DB2 Universal Database technical information" on page 377

Finding product information by accessing the DB2 Information Center from the administration tools

The DB2 Information Center provides quick access to DB2 product information and is available on all operating systems for which the DB2 administration tools are available.

The DB2 Information Center accessed from the tools provides six types of information.

Tasks Key tasks you can perform using DB2.

Concepts
Key concepts for DB2.

Reference
DB2 reference information, such as keywords, commands, and APIs.

Troubleshooting
Error messages and information to help you with common DB2 problems.

Samples
Links to HTML listings of the sample programs provided with DB2.

Tutorials
Instructional aid designed to help you learn a DB2 feature.

Prerequisites:

Some links in the DB2 Information Center point to Web sites on the Internet. To display the content for these links, you will first have to connect to the Internet.

Procedure:

To find product information by accessing the DB2 Information Center from the tools:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Click the tab of the information type related to the information you are attempting to find.
3. Navigate through the tree and click on the topic in which you are interested. The Information Center will then launch a Web browser to display the information.
4. To find information without browsing the lists, click the **Search** icon to the right of the list.

   Once the Information Center has launched a browser to display the information, you can perform a full-text search by clicking the **Search** icon in the navigation toolbar.

Related concepts:

- "Accessibility" on page 397
- "DB2 Information Center for topics" on page 399

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 388
- "Searching the DB2 documentation" on page 395

Viewing technical documentation online directly from the DB2 HTML Documentation CD

All of the HTML topics that you can install from the *DB2 HTML Documentation CD* can also be read directly from the CD. Therefore, you can view the documentation without having to install it.

Restrictions:

Because the following items are installed from the DB2 product CD and not the *DB2 HTML Documentation CD*, you must install the DB2 product to view these items:

- Tools help
- DB2 Quick Tour
- Release notes

Procedure:

1. Insert the *DB2 HTML Documentation* CD. On UNIX operating systems, mount the *DB2 HTML Documentation CD*. Refer to your *Quick Beginnings* book for details on how to mount a CD on UNIX operating systems.
2. Start your HTML browser and open the appropriate file:
   - For Windows operating systems:
     ```
     e:\Program Files\sqllib\doc\htmlcd\%L\index.htm
     ```
     where *e* represents the CD-ROM drive, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.
   - For UNIX operating systems:
     ```
     /cdrom/Program Files/sqllib/doc/htmlcd/%L/index.htm
     ```
     where */cdrom/* represents where the CD is mounted, and %L is the locale of the documentation that you wish to use, for example, **en_US** for English.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 388
- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 394

Related reference:

- "Overview of DB2 Universal Database technical information" on page 377

Updating the HTML documentation installed on your machine

It is now possible to update the HTML installed from the *DB2 HTML Documentation CD* when updates are made available from IBM. This can be done in one of two ways:

- Using the Information Center (if you have the DB2 administration GUI tools installed).
- By downloading and applying a DB2 HTML documentation FixPak .

Note: This will NOT update the DB2 code; it will only update the HTML documentation installed from the *DB2 HTML Documentation CD*.

Procedure:

To use the Information Center to update your local documentation:

1. Start the DB2 Information Center in one of the following ways:
   - From the graphical administration tools, click on the **Information Center** icon in the toolbar. You can also select it from the **Help** menu.
   - At the command line, enter **db2ic**.
2. Ensure your machine has access to the external Internet; the updater will download the latest documentation FixPak from the IBM server if required.
3. Select **Information Center —> Update Local Documentation** from the menu to start the update.
4. Supply your proxy information (if required) to connect to the external Internet.

The Information Center will download and apply the latest documentation FixPak, if one is available.

To manually download and apply the documentation FixPak :

1. Ensure your machine is connected to the Internet.
2. Open the DB2 support page in your Web browser at: www.ibm.com/software/data/db2/udb/winos2unix/support
3. Follow the link for version 8 and look for the ″Documentation FixPaks″ link.
4. Determine if the version of your local documentation is out of date by comparing the documentation FixPak level to the documentation level you have installed. This current documentation on your machine is at the following level: **DB2 v8.1 GA**.
5. If there is a more recent version of the documentation available then download the FixPak applicable to your operating system. There is one FixPak for all Windows platforms, and one FixPak for all UNIX platforms.
6. Apply the FixPak:
   - For Windows operating systems: The documentation FixPak is a self extracting zip file. Place the downloaded documentation FixPak in an empty directory, and run it. It will create a **setup** command which you can run to install the documentation FixPak.
   - For UNIX operating systems: The documentation FixPak is a compressed tar.Z file. Uncompress and untar the file. It will create a directory named `delta_install` with a script called **installdocfix**. Run this script to install the documentation FixPak.

Related tasks:

- "Copying files from the DB2 HTML Documentation CD to a Web Server" on page 394

Related reference:

- "Overview of DB2 Universal Database technical information" on page 377

Copying files from the DB2 HTML Documentation CD to a Web Server

The entire DB2 information library is delivered to you on the *DB2 HTML Documentation CD*, so you can install the library on a Web server for easier access. Simply copy to your Web server the documentation for the languages that you want.

Procedure:

To copy files from the *DB2 HTML Documentation CD* to a Web server, use the appropriate path:

- For Windows operating systems:

  ```
  E:\Program Files\sqllib\doc\htmlcd\%L\*.*
  ```

  where *E* represents the CD-ROM drive and *%L* represents the language identifier.
- For UNIX operating systems:

  ```
  /cdrom:Program Files/sqllib/doc/htmlcd/%L/*.*
  ```

  where *cdrom* represents the CD-ROM drive and *%L* represents the language identifier.

Related tasks:

- "Searching the DB2 documentation" on page 395

Related reference:

- "Supported DB2 interface languages, locales, and code pages" in the *Quick Beginnings for DB2 Servers*
- "Overview of DB2 Universal Database technical information" on page 377

Troubleshooting DB2 documentation search with Netscape 4.x

Most search problems are related to the Java support provided by web browsers. This task describes possible workarounds.

Procedure:

A common problem with Netscape 4.x involves a missing or misplaced security class. Try the following workaround, especially if you see the following line in the browser Java console:

```
Cannot find class  java/security/InvalidParameterException
```

- On Windows operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `x:Program Files\sqllib\doc\htmlcd\locale\InvalidParameterException.class` file to the `java\classes\java\security\` directory relative to your Netscape browser installation, where *x* represents the CD-ROM drive letter and *locale* represents the name of the desired locale.

  Note: You may have to create the `java\security\` subdirectory structure.
- On UNIX operating systems:

  From the *DB2 HTML Documentation CD*, copy the supplied `/cdrom/Program Files/sqllib/doc/htmlcd/locale/InvalidParameterException.class` file to the `java/classes/java/security/` directory relative to your Netscape browser installation, where *cdrom* represents the mount point of the CD-ROM and *locale* represents the name of the desired locale.

  Note: You may have to create the `java/security/` subdirectory structure.

If your Netscape browser still fails to display the search input window, try the following:
- Stop all instances of Netscape browsers to ensure that there is no Netscape code running on the machine. Then open a new instance of the Netscape browser and try to start the search again.
- Purge the browser's cache.
- Try a different version of Netscape, or a different browser.

Related tasks:
- "Searching the DB2 documentation" on page 395

Searching the DB2 documentation

To search DB2's documentation, you need Netscape 6.1 or higher, or Microsoft's Internet Explorer 5 or higher. Ensure that your browser's Java support is enabled.

A pop-up search window opens when you click the search icon in the navigation toolbar of the Information Center accessed from a browser. If you are using the search for the first time it may take a minute or so to load into the search window.

Restrictions:

The following restrictions apply when you use the documentation search:

- Boolean searches are not supported. The boolean search qualifiers *and* and *or* will be ignored in a search. For example, the following searches would produce the same results:
  - servlets *and* beans
  - servlets *or* beans
- Wildcard searches are not supported. A search on *java** will only look for the literal string *java** and would not, for example, find *javadoc*.

In general, you will get better search results if you search for phrases instead of single words.

Procedure:

To search the DB2 documentation:

1. In the navigation toolbar, click **Search**.
2. In the top text entry field of the Search window, enter two or more terms related to your area of interest and click **Search**. A list of topics ranked by accuracy displays in the **Results** field.

   Entering more terms increases the precision of your query while reducing the number of topics returned from your query.
3. In the **Results** field, click the title of the topic you want to read. The topic displays in the content frame.

Note: When you perform a search, the first result is automatically loaded into your browser frame. To view the contents of other search results, click on the result in results lists.

Related tasks:

- "Troubleshooting DB2 documentation search with Netscape 4.x" on page 394

Online DB2 troubleshooting information

With the release of DB2® UDB Version 8, there will no longer be a *Troubleshooting Guide*. The troubleshooting information once contained in this guide has been integrated into the DB2 publications. By doing this, we are able to deliver the most up-to-date information possible. To find information on the troubleshooting utilities and functions of DB2, access the DB2 Information Center from any of the tools.

Refer to the DB2 Online Support site if you are experiencing problems and want help finding possible causes and solutions. The support site contains a large, constantly updated database of DB2 publications, TechNotes, APAR (product problem) records, FixPaks, and other resources. You can use the support site to search through this knowledge base and find possible solutions to your problems.

Access the Online Support site at www.ibm.com/software/data/db2/udb/winos2unix/support, or by clicking the **Online Support** button in the DB2 Information Center. Frequently changing information, such as the listing of internal DB2 error codes, is now also available from this site.

Related concepts:

- "DB2 Information Center for topics" on page 399

Related tasks:

- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 390

Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in DB2® Universal Database Version 8:

- DB2 allows you to operate all features using the keyboard instead of the mouse. See "Keyboard Input and Navigation".
- DB2 enables you customize the size and color of your fonts. See "Accessible Display" on page 398.
- DB2 allows you to receive either visual or audio alert cues. See "Alternative Alert Cues" on page 398.
- DB2 supports accessibility applications that use the Java™ Accessibility API. See "Compatibility with Assistive Technologies" on page 398.
- DB2 comes with documentation that is provided in an accessible format. See "Accessible Documentation" on page 398.

Keyboard Input and Navigation

Keyboard Input

You can operate the DB2 Tools using only the keyboard. You can use keys or key combinations to perform most operations that can also be done using a mouse.

Keyboard Focus

In UNIX-based systems, the position of the keyboard focus is highlighted, indicating which area of the window is active and where your keystrokes will have an effect.

Accessible Display

The DB2 Tools have features that enhance the user interface and improve accessibility for users with low vision. These accessibility enhancements include support for customizable font properties.

Font Settings

The DB2 Tools allow you to select the color, size, and font for the text in menus and dialog windows, using the Tools Settings notebook.

Non-dependence on Color

You do not need to distinguish between colors in order to use any of the functions in this product.

Alternative Alert Cues

You can specify whether you want to receive alerts through audio or visual cues, using the Tools Settings notebook.

Compatibility with Assistive Technologies

The DB2 Tools interface supports the Java Accessibility API enabling use by screen readers and other assistive technologies used by people with disabilities.

Accessible Documentation

Documentation for the DB2 family of products is available in HTML format. This allows you to view documentation according to the display preferences set in your browser. It also allows you to use screen readers and other assistive technologies.

DB2 tutorials

The DB2® tutorials help you learn about various aspects of DB2 Universal Database. The tutorials provide lessons with step-by-step instructions in the areas of developing applications, tuning SQL query performance, working with data warehouses, managing metadata, and developing Web services using DB2.

Before you begin:

Before you can access these tutorials using the links below, you must install the tutorials from the *DB2 HTML Documentation* CD-ROM.

If you do not want to install the tutorials, you can view the HTML versions of the tutorials directly from the *DB2 HTML Documentation CD*. PDF versions of these tutorials are also available on the *DB2 PDF Documentation CD*.

Some tutorial lessons use sample data or code. See each individual tutorial for a description of any prerequisites for its specific tasks.

DB2 Universal Database tutorials:

If you installed the tutorials from the *DB2 HTML Documentation* CD-ROM, you can click on a tutorial title in the following list to view that tutorial.

Business Intelligence Tutorial: Introduction to the Data Warehouse Center
: Perform introductory data warehousing tasks using the Data Warehouse Center.

Business Intelligence Tutorial: Extended Lessons in Data Warehousing
: Perform advanced data warehousing tasks using the Data Warehouse Center. (Not provided on CD. You can download this tutorial from the Downloads section of the Business Intelligence Solutions Web site at http://www.ibm.com/software/data/bi/.)

Development Center Tutorial for Video Online using Microsoft® Visual Basic
: Build various components of an application using the Development Center Add-in for Microsoft Visual Basic.

Information Catalog Center Tutorial
: Create and manage an information catalog to locate and use metadata using the Information Catalog Center.

Video Central for e-business Tutorial
: Develop and deploy an advanced DB2 Web Services application using WebSphere® products.

Visual Explain Tutorial
: Analyze, optimize, and tune SQL statements for better performance using Visual Explain.

DB2 Information Center for topics

The DB2® Information Center gives you access to all of the information you need to take full advantage of DB2 Universal Database™ and DB2 Connect™ in your business. The DB2 Information Center also documents major DB2 features and components including replication, data warehousing, the Information Catalog Center, Life Sciences Data Connect, and DB2 extenders.

The DB2 Information Center accessed from a browser has the following features:

Regularly updated documentation
Keep your topics up-to-date by downloading updated HTML.

Search
Search all of the topics installed on your workstation by clicking **Search** in the navigation toolbar.

Integrated navigation tree
Locate any topic in the DB2 library from a single navigation tree. The navigation tree is organized by information type as follows:

- Tasks provide step-by-step instructions on how to complete a goal.
- Concepts provide an overview of a subject.
- Reference topics provide detailed information about a subject, including statement and command syntax, message help, requirements.

Master index
Access the information in topics and tools help from one master index. The index is organized in alphabetical order by index term.

Master glossary
The master glossary defines terms used in the DB2 Information Center. The glossary is organized in alphabetical order by glossary term.

Related tasks:

- "Finding topics by accessing the DB2 Information Center from a browser" on page 388
- "Finding product information by accessing the DB2 Information Center from the administration tools" on page 390
- "Updating the HTML documentation installed on your machine" on page 392

Appendix B. Notices for the DB2 Call Level Inteface Guide and Reference

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both, and have been used in at least one of the documents in the DB2 UDB documentation library.

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the documents in the DB2 UDB documentation library:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Contacting IBM

In the United States, call one of the following numbers to contact IBM:

- 1-800-237-5511 for customer service
- 1-888-426-4343 to learn about available service options
- 1-800-IBM-4YOU (426-4968) for DB2 marketing and sales

In Canada, call one of the following numbers to contact IBM:

- 1-800-IBM-SERV (1-800-426-7378) for customer service
- 1-800-465-9600 to learn about available service options
- 1-800-IBM-4YOU (1-800-426-4968) for DB2 marketing and sales

To locate an IBM office in your country or region, check IBM's Directory of Worldwide Contacts on the web at www.ibm.com/planetwide

Product information

Information regarding DB2 Universal Database products is available by telephone or by the World Wide Web at www.ibm.com/software/data/db2/udb

This site contains the latest information on the technical library, ordering books, client downloads, newsgroups, FixPaks, news, and links to web resources.

If you live in the U.S.A., then you can call one of the following numbers:

- 1-800-IBM-CALL (1-800-426-2255) to order products or to obtain general information.
- 1-800-879-2755 to order publications.

For information on how to contact IBM outside of the United States, go to the IBM Worldwide page at www.ibm.com/planetwide

Part Number: CT180NA

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC09-4849-00

(1P) P/N: CT180NA

Spine information:

IBM® DB2 Universal Database™ CLI Guide and Reference Version 8