

IBM DB2 10.1
for Linux, UNIX, and Windows

*Call Level Interface Guide and
Reference Volume 1
Updated January, 2013*



IBM DB2 10.1
for Linux, UNIX, and Windows

*Call Level Interface Guide and
Reference Volume 1
Updated January, 2013*



Note

Before using this information and the product it supports, read the general information under Appendix B, "Notices," on page 293.

Edition Notice

This document contains proprietary information of IBM. It is provided under a license agreement and is protected by copyright law. The information contained in this publication does not include any product warranties, and any statements provided in this manual should not be interpreted as such.

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

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

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

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

© Copyright IBM Corporation 2013.

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

Contents

About this book	vii	Binding parameter markers in CLI applications with column-wise array input	78
Chapter 1. Introduction to DB2 Call Level Interface and ODBC	1	Binding parameter markers in CLI applications with row-wise array input	79
Comparison of CLI and ODBC	2	Parameter diagnostic information in CLI applications	80
Chapter 2. IBM Data Server CLI and ODBC drivers	7	Changing parameter bindings in CLI applications with offsets	81
IBM Data Server Driver for ODBC and CLI overview	7	Specifying parameter values at execute time for long data manipulation in CLI applications.	82
Obtaining the IBM Data Server Driver for ODBC and CLI	8	Commit modes in CLI applications	83
Installing the IBM Data Server Driver for ODBC and CLI	8	When to call the CLI SQLEndTran() function	85
Configuring the IBM Data Server Driver for ODBC and CLI	11	Preparing and executing SQL statements in CLI applications	86
Connecting to databases for ODBC and CLI	20	Deferred prepare in CLI applications	87
Running CLI and ODBC applications using the IBM Data Server Driver for ODBC and CLI	34	Executing compound SQL (CLI) statements in CLI applications	88
Deploying the IBM Data Server Driver for ODBC and CLI with database applications	48	Cursors in CLI applications	89
Chapter 3. ODBC driver managers	51	Cursor considerations for CLI applications	92
unixODBC driver manager	51	Result set terminology in CLI applications	94
Setting up the unixODBC driver manager	51	Bookmarks in CLI applications	95
Microsoft ODBC driver manager	53	Rowset retrieval examples in CLI applications	96
DataDirect ODBC driver manager	53	Retrieving query results in CLI applications	97
Chapter 4. Initializing CLI applications	55	Column binding in CLI applications	99
Initialization and termination in CLI overview.	56	Specifying the rowset returned from the result set	100
Handles in CLI	57	Retrieving data with scrollable cursors in a CLI application	102
Chapter 5. Data types and data conversion in CLI applications	59	Retrieving data with bookmarks in a CLI application	104
String handling in CLI applications	60	Result set retrieval into arrays in CLI applications	106
Large object usage in CLI applications	62	Data retrieval in pieces in CLI applications	110
LOB locators in CLI applications	64	Fetching LOB data with LOB locators in CLI applications	111
Direct file input and output for LOB handling in CLI applications	66	XML data retrieval in CLI applications	112
LOB usage in ODBC applications	67	Inserting data	113
Long data for bulk inserts and updates in CLI applications	68	Inserting bulk data with bookmarks using SQLBulkOperations() in CLI applications	113
User-defined type (UDT) usage in CLI applications	69	Importing data with the CLI LOAD utility in CLI applications	114
Distinct type usage in CLI applications	70	XML column inserts and updates in CLI applications	117
XML data handling in CLI applications - Overview	71	Updating and deleting data in CLI applications	118
Changing of default XML type handling in CLI applications	72	Updating bulk data with bookmarks using SQLBulkOperations() in CLI applications	119
Chapter 6. Transaction processing in CLI overview	73	Deleting bulk data with bookmarks using SQLBulkOperations() in CLI applications	119
Allocating statement handles in CLI applications	74	Calling stored procedures from CLI applications	120
Issuing SQL statements in CLI applications	74	CLI stored procedure commit behavior	123
Parameter marker binding in CLI applications	75	Creating static SQL by using CLI/ODBC static profiling	123
Binding parameter markers in CLI applications	77	Capture file for CLI/ODBC/JDBC Static Profiling	126
		Considerations for mixing embedded SQL and CLI.	127

Freeing statement resources in CLI applications	128
Handle freeing in CLI applications	128

Chapter 7. Terminating a CLI application **131**

Chapter 8. Trusted connections through DB2 Connect. **133**

Creating and terminating a trusted connection through CLI.	134
Switching users on a trusted connection through CLI.	135

Chapter 9. Descriptors in CLI applications **139**

Consistency checks for descriptors in CLI applications	142
Descriptor allocation and freeing	143
Descriptor manipulation with descriptor handles in CLI applications	146
Descriptor manipulation without using descriptor handles in CLI applications.	147

Chapter 10. Catalog functions for querying system catalog information in CLI applications **149**

Input arguments on catalog functions in CLI applications	150
--	-----

Chapter 11. Programming hints and tips for CLI applications. **153**

Reduction of network flows with CLI array input chaining	159
--	-----

Chapter 12. Unicode CLI applications **161**

Unicode functions (CLI)	162
Unicode function calls to ODBC driver managers	163

Chapter 13. Multisite updates (two phase commit) in CLI applications . . . **165**

ConnectType CLI/ODBC configuration keyword	165
DB2 as transaction manager in CLI applications	166
Process-based XA-compliant Transaction Program Monitor (XA TP) programming considerations for CLI applications	168

Chapter 14. Asynchronous execution of CLI functions **171**

Executing functions asynchronously in CLI applications	172
--	-----

Chapter 15. Multithreaded CLI applications **175**

Application model for multithreaded CLI applications	176
Mixed multithreaded CLI applications	177

Chapter 16. Vendor escape clauses in CLI applications **179**

Extended scalar functions for CLI applications	182
--	-----

Chapter 17. Non-Java client support for high availability on IBM data servers **193**

Non-Java client support for high availability for connections to DB2 for Linux, UNIX, and Windows	194
Configuration of DB2 for Linux, UNIX, and Windows automatic client reroute support for non-Java clients	195
Example of enabling DB2 for Linux, UNIX, and Windows automatic client reroute support in non-Java clients	199
Configuration of DB2 for Linux, UNIX, and Windows workload balancing support for non-Java clients	201
Example of enabling DB2 for Linux, UNIX, and Windows workload balancing support in non-Java clients	202
Operation of the automatic client reroute feature when connecting to the DB2 for Linux, UNIX, and Windows server from an application other than a Java application	204
Operation of transaction-level workload balancing for connections to DB2 for Linux, UNIX, and Windows	206
Alternate groups for connections to DB2 Database for Linux, UNIX, and Windows from non-Java clients	207
Application programming requirements for high availability for connecting to DB2 for Linux, UNIX, and Windows servers	209
Client affinities for clients that connect to DB2 for Linux, UNIX, and Windows	210
Non-Java client support for high availability for connections to Informix servers	216
Configuration of Informix high-availability support for non-Java clients	217
Example of enabling Informix high availability support in non-Java clients	220
Operation of the automatic client reroute feature when connecting to the Informix database server from an application other than a Java application	221
Operation of workload balancing for connections to Informix from non-Java clients	223
Application programming requirements for high availability for connections from non-Java clients to Informix database server	224
Client affinities for connections to Informix database server from non-Java clients	224
Non-Java client support for high availability for connections to DB2 for z/OS servers	231
Configuration of Sysplex workload balancing and automatic client reroute for non-Java clients	233
Example of enabling DB2 for z/OS Sysplex workload balancing and automatic client reroute in non-Java client applications.	239

Operation of Sysplex workload balancing for connections from non-Java clients to DB2 for z/OS servers	240	Solaris CLI application compile and link options	264
Operation of the automatic client reroute feature for an application other than a Java application to the DB2 for z/OS server	241	Building CLI multi-connection applications on UNIX	265
Operation of transaction-level workload balancing for connections to the DB2 for z/OS data sharing group	244	Building CLI applications on Windows	267
Alternate groups for connections to DB2 for z/OS servers from non-Java clients	245	Windows CLI application compile and link options	268
Application programming requirements for the automatic client reroute environment when connecting to the DB2 for z/OS servers	247	Building CLI multi-connection applications on Windows	269
Chapter 18. XA support for a Sysplex in non-Java clients	249	Building CLI applications with configuration files	271
Enabling XA support for a Sysplex in non-Java clients	249	Building CLI stored procedures with configuration files	272
Chapter 19. Configuring your development environment to build and run CLI and ODBC applications	251	Chapter 21. Building CLI routines.	275
Setting up the ODBC environment (Linux and UNIX)	251	Building CLI routines on UNIX	275
Sample build scripts and configurations for the unixODBC driver manager	253	AIX CLI routine compile and link options	276
Setting up the Windows CLI environment	255	HP-UX CLI routine compile and link options	277
Selecting a different DB2 copy for your Windows CLI application	257	Linux CLI routine compile and link options	278
CLI bind files and package names	258	Solaris CLI routine compile and link options	279
Bind option limitations for CLI packages	260	Building CLI routines on Windows	280
Chapter 20. Building CLI applications	261	Windows CLI routine compile and link options	281
Building CLI applications on UNIX	261	Appendix A. Overview of the DB2 technical information	283
AIX CLI application compile and link options	262	DB2 technical library in hardcopy or PDF format	283
HP-UX CLI application compile and link options	262	Displaying SQL state help from the command line processor	286
Linux CLI application compile and link options	263	Accessing different versions of the DB2 Information Center	286
		Updating the DB2 Information Center installed on your computer or intranet server	286
		Manually updating the DB2 Information Center installed on your computer or intranet server	288
		DB2 tutorials	290
		DB2 troubleshooting information	290
		Terms and conditions	290
		Appendix B. Notices	293
		Index	297

About this book

The Call Level Interface (CLI) Guide and Reference is in two volumes. Volume 1 describes how to use CLI to create database applications for DB2® for Linux, UNIX, and Windows. Volume 2 is a reference that describes CLI functions, keywords and configuration.

The *Call Level Interface (CLI) Guide and Reference* is in two volumes:

- Volume 1 describes how to use CLI to create database applications for DB2 for Linux, UNIX, and Windows.
- Volume 2 is a reference that describes CLI functions, keywords and configuration.

Chapter 1. Introduction to DB2 Call Level Interface and ODBC

DB2 Call Level Interface (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.

You can use the CLI interface to access the following IBM® data server databases:

- DB2 Version 9 for Linux, UNIX, and Windows
- DB2 Universal Database™ Version 8 (and later) for OS/390® and z/OS®
- DB2 for IBM i 5.4 and later
- IBM Informix® database server Version 11.70

CLI is an alternative to embedded dynamic SQL, but unlike embedded SQL, it does not require host variables or a precompiler. Applications can be run against a variety of databases without having to be compiled against each of these databases. Applications use procedure calls at run time to connect to databases, issue SQL statements, and retrieve data and status information.

The CLI interface provides many features not available in embedded SQL. For example:

- CLI provides function calls that support a way of querying database catalogs that is consistent across the DB2 family. This reduces the need to write catalog queries that must be tailored to specific database servers.
- CLI provides the ability to scroll through a cursor:
 - Forward by one or more rows
 - Backward by one or more rows
 - Forward from the first row by one or more rows
 - Backward from the last row by one or more rows
 - From a previously stored location in the cursor.
- Stored procedures called from application programs that were written using CLI can return result sets to those programs.

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

CLI Background information

To understand 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. Because 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 through a mechanism called an escape clause.

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

The 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/downloads>. For non-Windows platforms, the ODBC SDK is provided by other vendors. When developing ODBC applications that may connect to DB2 servers, use the Call Level Interface Guide and Reference Volume 1 and the Call Level Interface Guide and Reference Volume 2 (for information about DB2 specific extensions and diagnostic information), in conjunction with the ODBC Programmer's Reference and SDK Guide available from Microsoft.

Applications written using CLI APIs link directly to the CLI library. 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 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)

Comparison of CLI and ODBC

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

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

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

Data Server Client refers to all available IBM Data Server Clients. DB2 server refers to all DB2 server products on Linux, UNIX, and Windows.

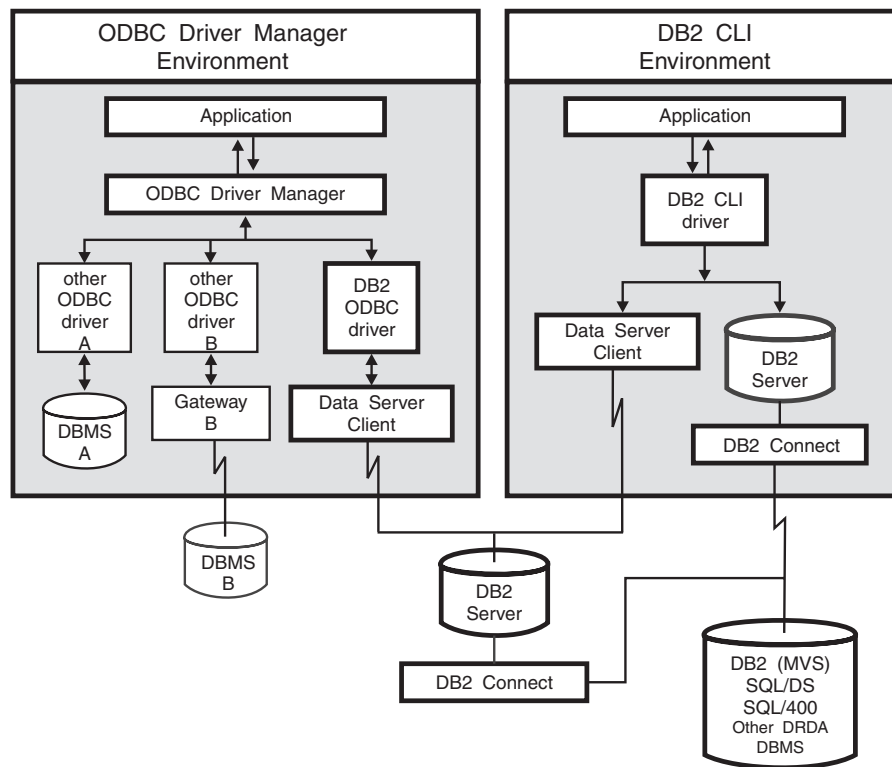


Figure 1. CLI and ODBC.

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 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/downloads>. For non-Windows platforms, the ODBC SDK is provided by other vendors.

In environments without an ODBC driver manager, 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. CLI ODBC support

ODBC features	DB2 ODBC Driver	CLI
Core level functions	All	All
Level 1 functions	All	All
Level 2 functions	All	All, except for SQLDrivers()
Additional CLI functions	All, functions can be accessed by dynamically loading the CLI library.	<ul style="list-style-type: none"> • SQLSetConnectAttr() • SQLGetEnvAttr() • SQLSetEnvAttr() • SQLSetColAttributes() • SQLGetSQLCA() • SQLBindFileToCol() • SQLBindFileToParam() • SQLExtendedBind() • SQLExtendedPrepare() • SQLGetLength() • SQLGetPosition() • SQLGetSubString()

Table 1. CLI ODBC support (continued)

ODBC features	DB2 ODBC Driver	CLI
SQL data types	All the types listed for CLI.	<ul style="list-style-type: none"> • 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_LONGVARBINARY • SQL_LONGVARCHAR • SQL_LONGVARGRAPHIC • SQL_NUMERIC • SQL_REAL • SQL_SMALLINT • SQL_TINYINT • SQL_TYPE_DATE • SQL_TYPE_TIME • SQL_TYPE_TIMESTAMP • SQL_VARBINARY • SQL_VARCHAR • SQL_VARGRAPHIC • SQL_WCHAR
C data types	All the types listed for CLI.	<ul style="list-style-type: none"> • SQL_C_BINARY • SQL_C_BIT • SQL_C_BLOB_LOCATOR • SQL_C_CHAR • SQL_C_CLOB_LOCATOR • SQL_C_TYPE_DATE • SQL_C_DBCHAR • SQL_C_DBCLOB_LOCATOR • SQL_C_DOUBLE • SQL_C_FLOAT • SQL_C_LONG • SQL_C_SHORT • SQL_C_TYPE_TIME • SQL_C_TYPE_TIMESTAMP • SQL_C_TIMESTAMP_EXT • SQL_C_TINYINT • SQL_C_SBIGINT • SQL_C_UBIGINT • SQL_C_NUMERIC ¹ • SQL_C_WCHAR
Return codes	All the codes listed for CLI.	<ul style="list-style-type: none"> • SQL_SUCCESS • SQL_SUCCESS_WITH_INFO • SQL_STILL_EXECUTING • SQL_NEED_DATA • SQL_NO_DATA_FOUND • SQL_ERROR • SQL_INVALID_HANDLE

Table 1. CLI ODBC support (continued)

ODBC features	DB2 ODBC Driver	CLI
SQLSTATES	Mapped to X/Open SQLSTATES with additional IBM SQLSTATES, with the exception of the ODBC type 08S01.	Mapped to X/Open SQLSTATES with additional IBM SQLSTATES
Multiple connections per application	Supported	Supported
Dynamic loading of driver	Supported	Not applicable

Note:

1. Only supported on Windows operating systems.
2. The listed SQL data types are supported for compatibility with ODBC 2.0.

- SQL_DATE
- SQL_TIME
- SQL_TIMESTAMP

You should use the SQL_TYPE_DATE, SQL_TYPE_TIME, or SQL_TYPE_TIMESTAMP instead to avoid any data type mappings.

3. The listed SQL data types and C data types are supported for compatibility with ODBC 2.0.

- SQL_C_DATE
- SQL_C_TIME
- SQL_C_TIMESTAMP

You should use the SQL_C_TYPE_DATE, SQL_C_TYPE_TIME, or SQL_C_TYPE_TIMESTAMP instead to avoid any data type mappings.

Isolation levels

The table 2 maps IBM RDBMSs 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 HY009 if you try to set an unsupported isolation level.	

Restriction

Mixing ODBC and CLI features and function calls in an application is not supported on the Windows 64-bit operating system.

Chapter 2. IBM Data Server CLI and ODBC drivers

In the IBM Data Server Client and the IBM Data Server Runtime Client there is a driver for the CLI application programming interface (API) and the ODBC API. This driver is commonly referred to throughout the DB2 Information Center and DB2 books as the IBM Data Server CLI driver or the IBM Data Server CLI/ODBC driver.

In the IBM Data Server Client and the IBM Data Server Runtime Client there is a driver for the CLI application programming interface (API) and the ODBC API. This driver is commonly referred to throughout the DB2 Information Center and DB2 books as the IBM Data Server CLI driver or the IBM Data Server CLI/ODBC driver.

New with DB2 Version 9, there is also a separate CLI and ODBC driver called the IBM Data Server Driver for ODBC and CLI. The IBM Data Server Driver for ODBC and CLI provides runtime support for the CLI and ODBC APIs. However, this driver is installed and configured separately, and supports a subset of the functionality of the DB2 clients, such as connectivity, in addition to the CLI and ODBC API support.

Information that applies to the CLI and ODBC driver that is part of the DB2 client generally applies to the IBM Data Server Driver for ODBC and CLI too. However, there are some restrictions and some functionality that is unique to the IBM Data Server Driver for ODBC and CLI. Information that applies only to the IBM Data Server Driver for ODBC and CLI will use the full title of the driver to distinguish it from general information that applies to the ODBC and CLI driver that comes with the DB2 clients.

- For more information about the IBM Data Server Driver for ODBC and CLI, see: "IBM Data Server Driver for ODBC and CLI overview."

IBM Data Server Driver for ODBC and CLI overview

The IBM Data Server Driver for ODBC and CLI provides runtime support for the CLI application programming interface (API) and the ODBC API. Though the IBM Data Server Client and IBM Data Server Runtime Client both support the CLI and ODBC APIs, this driver is not a part of either IBM Data Server Client or IBM Data Server Runtime Client. It is available separately, installed separately, and supports a subset of the functionality of the IBM Data Server Client.

The IBM Data Server Driver for ODBC and CLI provides runtime support for the CLI application programming interface (API) and the ODBC API. Though the IBM Data Server Client and IBM Data Server Runtime Client both support the CLI and ODBC APIs, this driver is not a part of either IBM Data Server Client or IBM Data Server Runtime Client. It is available separately, installed separately, and supports a subset of the functionality of the IBM Data Server Client.

Advantages of the IBM Data Server Driver for ODBC and CLI

- The driver has a much smaller footprint than the IBM Data Server Client and the IBM Data Server Runtime Client.
- You can have multiple installations of the driver on a single machine.

- You can install the driver on a machine that already has an IBM Data Server Client installed.
- You can include the driver in your database application installation package, and redistribute the driver with your applications. Under certain conditions, you can redistribute the driver with your database applications royalty-free.
- The driver can reside on an NFS mounted file system.

Functionality of the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI provides:

- runtime support for the CLI API;
 - runtime support for the ODBC API;
 - runtime support for the XA API;
 - database connectivity;
 - support for DB2 Interactive Call Level Interface (db2cli);
 - LDAP Database Directory support; and
 - tracing, logging, and diagnostic support.
- See: "Restrictions of the IBM Data Server Driver for ODBC and CLI" on page 37.

Obtaining the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It is available to download from the internet, and it is on the DB2 Version 9 install CD.

Procedure

You can obtain the IBM Data Server Driver for ODBC and CLI from following sources:

- Go to the IBM Support Fix Central website: <http://www-933.ibm.com/support/fixcentral/>. Data Server client and driver packages are found under the Information Management Product Group and IBM Data Server Client Packages Product selection. Select the appropriate Installed Version and Platform and click Continue. Click Continue again on the next screen and you will be presented with a list of all client and driver packages available for your platform, including IBM Data Server Driver for ODBC and CLI.

or

- Copy the driver from the DB2 install CD.

The driver is in a compressed file called

"ibm_data_server_driver_for_odbc_cli.zip" on Windows operating systems, and "ibm_data_server_driver_for_odbc_cli.tar.Z" on other operating systems.

Installing the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed separately.

Before you begin

To install the IBM Data Server Driver for ODBC and CLI, you need to obtain the compressed file that contains the driver. See "Obtaining the IBM Data Server Driver for ODBC and CLI."

About this task

There is no installation program for the IBM Data Server Driver for ODBC and CLI. You must install the driver manually.

Procedure

1. Copy the compressed file that contains the driver onto the target machine from the internet or a DB2 Version 9 installation CD.
2. Uncompress that file into your chosen install directory on the target machine.
3. Optional: Remove the compressed file.

Example

If you are installing the IBM Data Server Driver for ODBC and CLI under the following conditions:

- the operating systems on the target machine is AIX®; and
- the DB2 Version 9 CD is mounted on the target machine.

the steps you would follow are:

1. Create the directory `$HOME/db2_cli_odbc_driver`, where you will install the driver.
2. Locate the compressed file `ibm_data_server_driver_for_odbc_cli.tar.Z` on the install CD.
3. Copy `ibm_data_server_driver_for_odbc_cli.tar.Z` to the install directory, `$HOME/db2_cli_odbc_driver`.
4. Uncompress `ibm_data_server_driver_for_odbc_cli.tar.Z`:

```
cd $HOME/db2_cli_odbc_driver
uncompress ibm_data_server_driver_for_odbc_cli.tar.Z
tar -xvf ibm_data_server_driver_for_odbc_cli.tar
```
5. Delete `ibm_data_server_driver_for_odbc_cli.tar.Z`.
6. Ensure that the following requirements are met if you installed the driver on a NFS file system:
 - On UNIX or Linux operating systems the `db2dump` and the `db2` directory need to be writable. Alternatively, the path you have referenced in the **diagpath** parameter must be writable.
 - If host or i5/OS® data servers are being accessed directly ensure the `license` directory is writable.

Installing multiple copies of the IBM Data Server Driver for ODBC and CLI on the same machine

The IBM Data Server Driver for ODBC and CLI is not part of the IBM data server client or the IBM Data Server Runtime Client. It must be installed separately.

You can install multiple copies of the IBM Data Server Driver for ODBC and CLI on the same machine.

You might want to do this if you have two database applications on the same machine that require different versions of the driver.

Before you begin

To install multiple copies of the IBM Data Server Driver for ODBC and CLI on the same machine, you need to obtain the compressed file that contains the driver. See:

“Obtaining the IBM Data Server Driver for ODBC and CLI” on page 8.

Procedure

For each copy of the IBM Data Server Driver for ODBC and CLI that you are installing:

1. Create a unique target installation directory.
2. Follow the installation steps outlined in “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.
3. Ensure the application is using the correct copy of the driver. Avoid relying on the `LD_LIBRARY_PATH` environment variable as this can lead to inadvertent loading of the incorrect driver. Dynamically load the driver explicitly from the target installation directory.

Example

If you are installing two copies of the IBM Data Server Driver for ODBC and CLI under the following conditions:

- the operating systems on the target machine is AIX; and
- the DB2 Version 9 CD is mounted on the target machine.

the steps you would follow are:

1. Create the two directories, `$HOME/db2_cli_odbc_driver1` and `$HOME/db2_cli_odbc_driver2`, where you will install the driver.
2. Locate the compressed file that contains the driver on the install CD. In this scenario, the file would be called `ibm_data_server_driver_for_odbc_cli.tar.Z`.
3. Copy `ibm_data_server_driver_for_odbc_cli.tar.Z` to the install directories, `$HOME/db2_cli_odbc_driver1` and `$HOME/db2_cli_odbc_driver2`.
4. Uncompress `ibm_data_server_driver_for_odbc_cli.tar.Z` in each directory:

```
cd $HOME/db2_cli_odbc_driver1
uncompress ibm_data_server_driver_for_odbc_cli.tar.Z
tar -xvf ibm_data_server_driver_for_odbc_cli.tar
cd $HOME/db2_cli_odbc_driver2
uncompress ibm_data_server_driver_for_odbc_cli.tar.Z
tar -xvf ibm_data_server_driver_for_odbc_cli.tar
```
5. Delete `ibm_data_server_driver_for_odbc_cli.tar.Z`.

Installing the IBM Data Server Driver for ODBC and CLI on a machine with an existing DB2 client

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed separately.

You can install one or more copies of the IBM Data Server Driver for ODBC and CLI on a machine where an IBM Data Server Client or IBM Data Server Runtime Client is already installed. You might want to do this if you have developed some ODBC or CLI database applications with the IBM Data Server Client that you plan to deploy with the IBM Data Server Driver for ODBC and CLI, because it enables you to test the database applications with the driver on the same machine as your development environment.

Before you begin

To install the IBM Data Server Driver for ODBC and CLI on the same machine as an IBM Data Server Client or IBM Data Server Runtime Client, you need:

- to obtain the compressed file that contains the driver.
 - See: “Obtaining the IBM Data Server Driver for ODBC and CLI” on page 8.

About this task

The procedure for installing one or more copies of the IBM Data Server Driver for ODBC and CLI on a machine that already has an IBM Data Server Client or IBM Data Server Runtime Client installed is the same as the procedure for installing the driver on a machine that has no IBM Data Server Client installed.

Procedure

See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8 and “Installing multiple copies of the IBM Data Server Driver for ODBC and CLI on the same machine” on page 9.

What to do next

Ensure the application is using the correct copy of the driver. Avoid relying on the `LD_LIBRARY_PATH` environment variable as this can lead to inadvertent loading of the incorrect driver. Dynamically load the driver explicitly from the target installation directory.

Configuring the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately. You must configure the IBM Data Server Driver for ODBC and CLI, and the software components of your database application runtime environment in order for your applications to use the driver successfully.

Before you begin

To configure the IBM Data Server Driver for ODBC and CLI and your application environment for the driver, you need:

- one or more copies of the driver installed.
 - See “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.

Procedure

To configure the IBM Data Server Driver for ODBC and CLI, and the runtime environment of your IBM Data Server Driver for ODBC and CLI applications to use the driver:

1. Configure aspects of the driver's behavior such as data source name, user name, performance options, and connection options by updating the `db2cli.ini` initialization file.

The location of the `db2cli.ini` file might change based on whether the Microsoft ODBC Driver Manager is used, the type of data source names (DSN) used, the type of client or driver being installed, and whether the registry variable `DB2CLIINIPATH` is set.

- See “db2cli.ini initialization file” on page 12.

There is no support for the Command Line Processor (CLP) with the IBM Data Server Driver for ODBC and CLI. For this reason, you can not update CLI configuration using the CLP command "db2 update CLI cfg"; you must update the `db2cli.ini` initialization file manually.

If you have multiple copies of the IBM Data Server Driver for ODBC and CLI installed, each copy of the driver will have its own `db2cli.ini` file. Ensure you make the additions to the `db2cli.ini` for the correct copy of the driver.

2. Configure application environment variables.
 - See "Configuring environment variables for the IBM Data Server Driver for ODBC and CLI" on page 15.
3. For applications participating in transactions managed by the Microsoft Distributed Transaction Coordinator (DTC) only, you must register the driver with the DTC.
 - See "Registering the IBM Data Server Driver for ODBC and CLI with the Microsoft DTC" on page 18.
4. For ODBC applications using the Microsoft ODBC driver manager only, you must register the driver with the Microsoft driver manager.
 - See "Registering the IBM Data Server Driver for ODBC and CLI with the Microsoft ODBC driver manager" on page 19.

db2cli.ini initialization file

The CLI/ODBC initialization file (`db2cli.ini`) contains various keywords and values that can be used to configure the behavior of CLI and the applications using it.

The keywords are associated with the database alias name, and affect all CLI and ODBC applications that access the database.

The `db2cli.ini.sample` sample configuration file is included to help you get started. You can create a `db2cli.ini` file that is based on the `db2cli.ini.sample` file and that is stored in the same location. The location of the sample configuration file depends on your driver type and platform.

For IBM Data Server Client, IBM Data Server Runtime Client, or IBM Data Server Driver Package, the sample configuration file is created in one of the following paths:

- On AIX, HP-UX, Linux, or Solaris operating systems: *installation_path/cfg*
- On Windows XP and Windows Server 2003: `C:\Documents and Settings\All Users\Application Data\IBM\DB2\driver_copy_name\cfg`
- On Windows Vista and Windows Server 2008: `C:\ProgramData\IBM\DB2\driver_copy_name\cfg`

For example, if you use IBM Data Server Driver Package for Windows XP, and the data server driver copy name is `IBMDBCL1`, then the `db2cli.ini.sample` file is created in the `C:\Documents and Settings\All Users\Application Data\IBM\DB2\IBMDBCL1\cfg` directory.

For IBM Data Server Driver for ODBC and CLI, the sample configuration file is created in one of the following paths:

- On AIX, HP-UX, Linux, or Solaris operating systems: *installation_path/cfg*
- On Windows : *installation_path\cfg*
where *installation_path* is the file path where driver files are extracted.

For example, if you use IBM Data Server Driver for ODBC and CLI for Windows Vista, and the driver is installed in the C:\IBMDB2\CLIDRIVER\V97FP3 directory, then the db2cli.ini.sample file is created in the C:\IBMDB2\CLIDRIVER\V97FP3\cfg directory.

When the ODBC Driver Manager is used to configure a user DSN on Windows operating systems, the db2cli.ini file is created in Documents and Settings\User Name where *User Name* represents the name of the user directory.

You can use the environment variable **DB2CLIINIPATH** to specify a different location for the db2cli.ini 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.
- Override default connection options specified by an application. For example, if an application requests Unicode support from the CLI driver by setting the SQL_ATTR_ANSI_APP connection attribute, then setting **DisableUnicode=1** in the db2cli.ini file will force the CLI driver not to provide the application with Unicode support.

Note: If the CLI/ODBC configuration keywords set in the db2cli.ini file conflict with keywords in the SQLDriverConnect() connection string, then the SQLDriverConnect() keywords will take precedence.

The db2cli.ini initialization file is an ASCII file which stores values for the CLI configuration options. A sample file is included to help you get started. While most CLI/ODBC configuration keywords are set in the db2cli.ini initialization file, some keywords are set by providing the keyword information in the connection string to SQLDriverConnect() instead.

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 database connections.

Only the keywords that apply to all database connections through the CLI/ODBC driver are included in the COMMON section. This includes the following keywords:

- **CheckForFork**
- **DiagPath**
- **DisableMultiThread**
- **JDBCTrace**
- **JDBCTraceFlush**
- **JDBCTracePathName**
- **QueryTimeoutInterval**
- **ReadCommonSectionOnNullConnect**
- **Trace**
- **TraceComm**

- **TraceErrImmediate**
- **TraceFileName**
- **TraceFlush**
- **TraceFlushOnError**
- **TraceLocks**
- **TracePathName**
- **TracePIDList**
- **TracePIDTID**
- **TraceRefreshInterval**
- **TraceStmtOnly**
- **TraceTime**
- **TraceTimeStamp**

All other keywords are to be placed in the database specific section.

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 affect on all CLI/ODBC connections from that client. A keyword such as **TRACE**, for example, will generate information about all 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 under 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 sample .INI file contains two database alias sections:


```

; This is a comment line.
[MYDB22]
AutoCommit=0
TableType="'TABLE','SYSTEM TABLE'"

; This is another comment line.
[MYDB2MVS]
CurrentSQLID=SAIID
TableType="'TABLE'"
SchemaList="'USER1',CURRENT SQLID,'USER2'"

```

Although you can edit the `db2cli.ini` file manually on all platforms, it is recommended that you use the **UPDATE CLI CONFIGURATION** command if it is available. You must add a blank line after the last entry if you manually edit the `db2cli.ini` file.

Configuring environment variables for the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately.

To use the IBM Data Server Driver for ODBC and CLI, there are two types of environment variables that you might have to set: environment variables that have replaced some DB2 registry variables; and an environment variable that tells your applications where to find the driver libraries.

Before you begin

To configure environment variables for the IBM Data Server Driver for ODBC and CLI, you need one or more copies of the driver installed. See “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.

Restrictions

If there are multiple versions of the IBM Data Server Driver for ODBC and CLI installed on the same machine, or if there are other DB2 Version 9 products installed on the same machine, setting environment variables (for example, setting **LIBPATH** or **LD_LIBRARY_PATH** to point to the IBM Data Server Driver for ODBC and CLI library) might break existing applications. When setting an environment variable, ensure that it is appropriate for all applications running in the scope of that environment.

IBM Data Server Driver for ODBC and CLI on 64 bit UNIX and Linux systems also packages 32 bit driver library to support 32 bit CLI applications. On UNIX and Linux systems, you can associate either 32 bit or 64 bit libraries with an instance, not both. You can set **LIBPATH** or **LD_LIBRARY_PATH** to either `lib32` or `lib64` library directories of IBM Data Server Driver for ODBC and CLI. You can also access `lib64` library using a preset soft link from the `lib` directory. The IBM Data Server Driver for ODBC and CLI on Windows systems contains both 32 bit and 64 bit necessary runtime DLLs in the same `bin` directory.

Procedure

To configure environment variables for the IBM Data Server Driver for ODBC and CLI:

1. Optional: Set any applicable DB2 environment variables corresponding to its equivalent DB2 registry variables.

There is no support for the command line processor (CLP) with the IBM Data Server Driver for ODBC and CLI. For this reason, you cannot configure DB2 registry variables using the **db2set** CLP command. Required DB2 registry variables have been replaced with environment variables.

For a list of the environment variables that can be used instead of DB2 registry variables, see: "Environment variables supported by the IBM Data Server Driver for ODBC and CLI."

2. Optional: You can set the local environment variable **DB2_CLI_DRIVER_INSTALL_PATH** to the directory in which the driver is installed. If there are multiple copies of the IBM Data Server Driver for ODBC and CLI installed, ensure that the **DB2_CLI_DRIVER_INSTALL_PATH** points to the intended copy of the driver. Setting the **DB2_CLI_DRIVER_INSTALL_PATH** variable forces IBM Data Server Driver for ODBC and CLI to use the directory specified with the **DB2_CLI_DRIVER_INSTALL_PATH** variable as the install location of the driver. For example:

```
export DB2_CLI_DRIVER_INSTALL_PATH=/home/db2inst1/db2clidriver/clidriver
```

where /home/db2inst1/db2clidriver is the install path where the CLI driver is installed

3. Optional: Set the environment variable **LIBPATH** (on AIX operating systems), **SHLIB_PATH** (on HP-UX systems), or **LD_LIBRARY_PATH** (on rest of the UNIX and Linux systems) to the lib directory in which the driver is installed. For example (on AIX systems):

```
export LIBPATH=/home/db2inst1/db2clidriver/clidriver/lib
```

If there are multiple copies of the IBM Data Server Driver for ODBC and CLI installed, ensure **LIBPATH** or **LD_LIBRARY_PATH** points to the intended copy of the driver. Do not set **LIBPATH** or **LD_LIBRARY_PATH** variables to multiple copies of the IBM Data Server Driver for ODBC and CLI that are installed on your system. Do not set **LIBPATH** or **LD_LIBRARY_PATH** variables to both lib32 and lib64 (or lib) library directories.

This step is not necessary if your applications statically link to, or dynamically load the driver library (db2cli.dll on Windows systems, or libdb2.a on other systems) with the fully qualified name.

You must dynamically load the library using the fully qualified library name. On Windows operating systems, you must use the LoadLibraryEx method specifying the **LOAD_WITH_ALTERED_SEARCH_PATH** parameter and the path to the driver DLL.

4. Optional: Set the **PATH** environment variable to include the bin directory of the driver installation in all systems if you require the use of utilities like **db2level**, **db2cli**, and **db2trc**. In UNIX and Linux systems, add adm directory of the driver installation to the **PATH** environment variable in addition to the bin directory. For Example (on all UNIX and Linux systems) :

```
export PATH=/home/db2inst1/db2clidriver/clidriver/bin:/home/db2inst1/db2clidriver/clidriver/adm:$PATH
```

Environment variables supported by the IBM Data Server Driver for ODBC and CLI:

The IBM Data Server Driver for ODBC and CLI does not support the command line processor (CLP). This means that the usual mechanism to set DB2 registry

variables, using the **db2set** CLP command, is not possible. Relevant DB2 registry variables will be supported with the IBM Data Server Driver for ODBC and CLI as environment variables instead.

IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately.

The IBM Data Server Driver for ODBC and CLI does not support the command line processor (CLP). This means that the usual mechanism to set DB2 registry variables, using the **db2set** CLP command, is not possible. Relevant DB2 registry variables will be supported with the IBM Data Server Driver for ODBC and CLI as environment variables instead.

The DB2 registry variables that will be supported by the IBM Data Server Driver for ODBC and CLI as environment variables are:

Table 3. DB2 registry variables supported as environment variables

Type of variable	Variable name(s)
General variables	DB2ACCOUNT DB2BIDI DB2CODEPAGE DB2GRAPHICUNICODESERVER DB2LOCALE DB2TERRITORY
System environment variables	DB2DOMAINLIST
Communications variables	DB2_FORCE-NLS_CACHE DB2SORCVBUF DB2SOSNDBUF DB2TCP_CLIENT_RCVTIMEOUT
Performance variables	DB2_NO_FORK_CHECK
Miscellaneous variables	DB2CLINIPATH DB2SDRIVER_CFG_PATH DB2SDRIVER_CLIENT_HOSTNAME DB2_ENABLE_LDAP DB2LDAP_BASEDN DB2LDAP_CLIENT_PROVIDER DB2LDAPHOST DB2LDAP_KEEP_CONNECTION DB2LDAP_SEARCH_SCOPE DB2NOEXITLIST
Diagnostic variables	DB2_DIAGPATH
Connection variables	AUTHENTICATION PROTOCOL PWDPLUGIN KRBPLUGIN ALHOSTNAME ALTPORT INSTANCE BIDI

db2oreg1.exe overview

You can use the db2oreg1.exe utility to register the XA library of the IBM Data Server Driver for ODBC and CLI with the Microsoft Distributed Transaction Coordinator (DTC), and to register the driver with the Microsoft ODBC driver manager.

You need to use the db2oreg1.exe utility on Windows operating systems only.

The db2oreg1.exe utility is deprecated and will become unavailable in a future release. Use the **db2cli** DB2 interactive CLI command instead.

For IBM Data Server client packages on Windows 64-bit operating systems, the 32-bit version of the db2oreg1.exe utility (db2oreg132.exe) is supported in addition to the 64-bit version of db2oreg1.exe.

Conditions requiring that you run the db2oreg1.exe utility

You must run the db2oreg1.exe utility if:

- your applications that use the IBM Data Server Driver for ODBC and CLI will be participating in distributed transactions managed by the DTC; or
- your applications that use the IBM Data Server Driver for ODBC and CLI will be connecting to ODBC data sources.

You can also run the db2oreg1.exe utility to create the db2dsdriver.cfg.sample and db2cli.ini.sample sample configuration files.

When to run the db2oreg1.exe utility

If you use the db2oreg1.exe utility, you must run it when:

- you install the IBM Data Server Driver for ODBC and CLI; and
- you uninstall the IBM Data Server Driver for ODBC and CLI.

The db2oreg1.exe utility makes changes to the Windows registry when you run it after installing the driver. If you uninstall the driver, you should run the utility again to undo those changes.

How to run the db2oreg1.exe utility

- db2oreg1.exe is located in bin subdirectory where the IBM Data Server Driver for ODBC and CLI is installed.
- To list the parameters the db2oreg1.exe utility takes, and how to use them, run the utility with the "-h" option.

Registering the IBM Data Server Driver for ODBC and CLI with the Microsoft DTC

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately.

To use the IBM Data Server Driver for ODBC and CLI with database applications that participate in transactions managed by the Microsoft Distributed Transaction Coordinator (DTC), you must register the driver with the DTC.

Here is a link to the Microsoft article outlining the details of this security requirement: [Registry Entries Are Required for XA Transaction Support](#)

Before you begin

To register the IBM Data Server Driver for ODBC and CLI with the DTC, you need one or more copies of the driver installed. See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.

Restrictions

You only need to register the IBM Data Server Driver for ODBC and CLI with the DTC if your applications that use the driver are participating in transactions managed by the DTC.

Procedure

To register the IBM Data Server Driver for ODBC and CLI with the DTC, run the `db2cli install -setup` command for each copy of the driver that is installed: The `db2cli install -setup` command makes changes to the Windows registry when you run it after installing the driver. If you uninstall the driver, you should run the `db2cli install -cleanup` command to undo those changes.

Example

For example, the following command registers the IBM Data Server Driver for ODBC and CLI in the Windows registry, and creates the configuration folders under the application data path:

```
> db2cli install -setup
```

The IBM Data Server Driver for ODBC and CLI registered successfully.
The configuration folders are created successfully.

Registering the IBM Data Server Driver for ODBC and CLI with the Microsoft ODBC driver manager

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately. For ODBC applications to use the IBM Data Server Driver for ODBC and CLI with the Microsoft ODBC driver manager, you must register the driver with the driver manager.

Before you begin

To register the IBM Data Server Driver for ODBC and CLI with the Microsoft ODBC driver manager, you need one or more copies of the driver installed. See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.

About this task

The Microsoft ODBC driver manager is the only ODBC driver manager with which you must register the IBM Data Server Driver for ODBC and CLI. The other ODBC driver managers do not require this activity.

Procedure

To register the IBM Data Server Driver for ODBC and CLI with the Microsoft driver manager, run the `db2cli install -setup` command for each copy of the driver that is installed.

Results

The **db2cli install -setup** command changes the Windows registry when you run it after installing the driver. If you uninstall the driver, run the **db2cli install -cleanup** command to undo those changes.

Example

For example, the following command registers the IBM Data Server Driver for ODBC and CLI in the Windows registry, and creates the configuration folders under the application data path:

```
> db2cli install -setup
```

The IBM Data Server Driver for ODBC and CLI registered successfully.
The configuration folders are created successfully.

Connecting to databases for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately.

The IBM Data Server Driver for ODBC and CLI does not create a local database directory. This means that when you use this driver, you must make connectivity information available to your applications in other ways.

Before you begin

To connect to databases with the IBM Data Server Driver for ODBC and CLI, you need:

- Databases to which to connect; and
- One or more copies of the driver installed.
 - For more information, see “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.

About this task

There are several ways to specify connectivity information so that your CLI and ODBC database applications can use the IBM Data Server Driver for ODBC and CLI to connect to a database. When CLI settings are specified in multiple places, they are used in the listed order:

1. Connection strings parameters
2. db2cli.ini file
3. db2dsdriver.cfg file

Procedure

To configure connectivity for a database when using the IBM Data Server Driver for ODBC and CLI, use one of the listed methods:

- Specify the database connectivity information in the connection string parameter to `SQLDriverConnect`.
 - For more information, see “SQLDriverConnect function (CLI) - (Expanded) Connect to a data source” on page 25.
- For CLI applications only: put the database connectivity information in the CLI configuration file.

There is no support for the Command Line Processor (CLP) with the IBM Data Server Driver for ODBC and CLI. For this reason, you cannot update CLI configuration by using the CLP command "db2 update CLI cfg"; you must update the db2cli.ini initialization file manually.

If you have multiple copies of the IBM Data Server Driver for ODBC and CLI installed, each copy of the driver has its own db2cli.ini file. Ensure that you make the additions to the db2cli.ini for the correct copy of the driver.

For more information about the location of the db2cli.ini file, see "db2cli.ini initialization file" on page 12.

- Use the db2dsdriver.cfg configuration file to provide connection information and parameters. For example, you can specify the listed information in the db2dsdriver.cfg configuration file, and pass the connection string in SQLDriverConnect() as DSN=myDSN;PWD=XXXXX:

```
<configuration>
  <dsncollection>
    <dsn alias="myDSN" name="sample" host="server.domain.com" port="446">
    </dsn>
  </dsncollection>
  <databases>
    <database name="sample" host="server.domain.com" port="446">
      <parameter name="CommProtocol" value="TCPIP"/>
      <parameter name="UID" value="username"/>
    </database>
  </databases>
</configuration>
```

- Starting with DB2 Version 10.1 Fix Pack 2 and later fix packs, you can register the ODBC data source names (DSN) during silent installation on Windows platforms.

Specify the DB2_ODBC_DSN_TYPE and DB2_ODBC_DSN_ACTION keywords in the installation response file to register the ODBC DSNs. You can specify the type of DSN and also provide a complete list of DSNs.

DSNs are registered during silent installation for the current copy name only. Installation does not update ODBC DSN registries of any other copy available on the same client system.

For a 32-bit client package, installation creates ODBC DSNs under 32-bit registry hierarchy. For a 64-bit client package, installation creates ODBC DSNs under both the 32-bit and 64-bit registry hierarchy. However, for USER DSNs in a 64-bit client package, because Windows maintains a common ODBC DSN registry entry for both 32-bit and 64-bit product environments, installation modifies only the 64-bit user DSNs.

Note: The Windows platform maintains different registry paths to track 32-bit and 64-bit product environment settings for SYSTEM DSNs. So this restriction is not applicable for SYSTEM DSNs and installation adds system DSNs to both 32-bit and 64-bit registry hierarchy.

- For ODBC applications only: register the database as an ODBC data source with the ODBC driver manager. For more information, see "Registering ODBC data sources for applications that use the IBM Data Server Driver for ODBC and CLI" on page 23.
- Use the **FileDSN** CLI/ODBC keyword to identify a file data source name (DSN) that contains the database connectivity information. For more information, see "FileDSN CLI/ODBC configuration keyword" on page 32.

A file DSN is a file that contains database connectivity information. You can create a file DSN by using the **SaveFile** CLI/ODBC keyword. On Windows operating systems, you can use the Microsoft ODBC driver manager to create a file DSN.

- For local database servers only: use the **PROTOCOL** and the **INSTANCE** CLI/ODBC keywords to identify the local database.
 1. Set the **PROTOCOL** CLI/ODBC keyword to the value Local.
 2. Set the **INSTANCE** CLI/ODBC keyword to the instance name of the local database server on which the database is located.

For more information, see “Protocol CLI/ODBC configuration keyword” on page 33 and “Instance CLI/ODBC configuration keyword” on page 32.

Example

Here is a list of CLI/ODBC keywords that work with file DSN or DSN-less connections:

- “AltHostName CLI/ODBC configuration keyword” on page 30;
- “AltPort CLI/ODBC configuration keyword” on page 31;
- “Authentication CLI/ODBC configuration keyword” on page 31;
- “BIDI CLI/ODBC configuration keyword” on page 32;
- “FileDSN CLI/ODBC configuration keyword” on page 32;
- “Instance CLI/ODBC configuration keyword” on page 32;
- “Interrupt CLI/ODBC configuration keyword” on page 33;
- “KRBPlugin CLI/ODBC configuration keyword” on page 33;
- “Protocol CLI/ODBC configuration keyword” on page 33;
- “PWDPlugin CLI/ODBC configuration keyword” on page 34;
- “SaveFile CLI/ODBC configuration keyword” on page 34;
- “DiagLevel CLI/ODBC configuration keyword” on page 40;
- “NotifyLevel CLI/ODBC configuration keyword” on page 40;

For the examples, consider a database with the listed properties:

- The database or subsystem is called db1 on the server
- The server is located at 11.22.33.44
- The access port is 56789
- The transfer protocol is TCP/IP.

To make a connection to the database in a CLI application, you can perform one of the listed actions:

- Call `SQLDriverConnect` with a connection string that contains: `Database=db1; Protocol=tcip; Hostname=11.22.33.44; Servicename=56789;`
- Add the example to `db2cli.ini`:

```
[db1]
Database=db1
Protocol=tcip
Hostname=11.22.33.44
Servicename=56789
```

To make a connection to the database in an ODBC application:

1. Register the database as an ODBC data source called `odbc_db1` with the driver manager.

2. Call `SQLConnect` with a connection string that contains: `Database=odbc_db1`;

Registering ODBC data sources for applications that use the IBM Data Server Driver for ODBC and CLI

You must install and configure the IBM Data Server Driver for ODBC and CLI before an ODBC database application can use the driver. This driver is not part of the IBM Data Server Client or the IBM Data Server Runtime Client.

Before you begin

To register a database as an ODBC data source and associate the IBM Data Server Driver for ODBC and CLI with the database, listed requirements must be met:

- Databases to which your ODBC applications connect
- An ODBC driver manager installed
- One or more copies of the IBM Data Server Driver for ODBC and CLI installed
 - See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.
- one or more copies of the driver installed.
 - See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.

About this task

The name of the IBM Data Server Driver for ODBC and CLI library file is `db2app.dll` on Windows operating systems, and `db2app.lib` on other platforms. The driver library file is located in the `lib` subdirectory of the directory in which you installed the driver.

If you have multiple copies of the IBM Data Server Driver for ODBC and CLI installed, ensure that the intended copy is identified in the `odbc.ini` file. When possible, avoid installing multiple copies of this driver.

Procedure

This procedure depends on which driver manager you are using for your applications.

- For the Microsoft ODBC driver manager, perform the listed actions:
 1. Register the IBM Data Server Driver for ODBC and CLI with the Microsoft ODBC driver manager by using the `db2cli install -setup` command. See “Registering the IBM Data Server Driver for ODBC and CLI with the Microsoft ODBC driver manager” on page 19.
 2. Register the database as an ODBC data source. See “Setting up the Windows CLI environment” on page 255.
- For open source ODBC driver managers, perform the listed actions:
 1. Identify the database as an ODBC data source by adding database information to the `odbc.ini` file. See “Setting up the ODBC environment (Linux and UNIX)” on page 251.
 2. Associate the IBM Data Server Driver for ODBC and CLI with the data source by adding it in the database section of the `odbc.ini` file. You must use the fully qualified library file name.

Results

Whenever you create a Microsoft ODBC data source by using the ODBC Driver Manager, you must manually open the ODBC data source Administrator and

create a data source by checking the contents of the `db2cli.ini` file or the `db2dsdriver.cfg` file. There is no command-line utility that reads the `db2cli.ini` file or the `db2dsdriver.cfg` file and creates a Microsoft ODBC data source. The **db2cli** command provides additional options to create a Microsoft ODBC data source through the **registerdsn** command parameter, which offers the listed functions:

- Registers a Microsoft system or user ODBC data source if a data source entry is available in the `db2cli.ini` or `db2dsdriver.cfg` file or in the local database directory as a cataloged database.
- Lists all the DB2 system or user data sources that are already registered in the Microsoft Data Source Administrator.
- Removes the system or user data sources that are already registered in the Microsoft Data Source Administrator.

Note: The **db2cli registerdsn** command is supported only on Microsoft Windows operating systems.

Example

You want to register ODBC data sources with an open source driver manager under the listed conditions:

- The operating system for the target database server is AIX.
- There are two copies of the IBM Data Server Driver for ODBC and CLI installed at
 - `$HOME/db2_cli_odbc_driver1` and
 - `$HOME/db2_cli_odbc_driver2`
- You have two ODBC database applications:
 - ODBCapp_A
 - ODBCapp_A connects to two data sources, db1 and db2
 - The application should use the copy of the driver installed at `$HOME/db2_cli_odbc_driver1`.
 - ODBCapp_B
 - ODBCapp_B connects to the data source db3
 - The application should use the copy of the driver installed at `$HOME/db2_cli_odbc_driver2`.

To register ODBC data sources with an open source driver manager, add the example entries in the `odbc.ini` file:

```
[db1]
Driver=$HOME/db2_cli_odbc_driver1/lib/libdb2.a
Description=First ODBC data source for ODBCapp1,
    using the first copy of the IBM Data Server Driver for ODBC and CLI

[db2]
Driver=$HOME/db2_cli_odbc_driver1/lib/libdb.a
Description=Second ODBC data source for ODBCapp1,
    using the first copy of the IBM Data Server Driver for ODBC and CLI

[db3]
Driver=$HOME/db2_cli_odbc_driver2/lib/libdb2.a
Description=First ODBC data source for ODBCapp2,
    using the second copy of the IBM Data Server Driver for ODBC and CLI
```

Every time when a Microsoft ODBC Data Source (using ODBC Driver Manager) has to be created, user has to manually open the ODBC Data Source Administrator and create a Data Source by checking the contents of `db2cli.ini` or the `db2dsdriver.cfg` files. There is no command line utility which reads the `db2cli.ini` file or the `db2dsdriver.cfg` file and creates a Microsoft ODBC Data Source. Hence to overcome this concern, the **db2cli** provides additional options to create a Microsoft ODBC Data Source through the new **registerdsn** command parameter which offers the listed functions:

- Register a Microsoft System/User ODBC Data Source if a Data Source entry is available in `db2cli.ini` or in `db2dsdriver.cfg` file
- Register all the Data Sources available in the `db2cli.ini` file or the `db2dsdriver.cfg` file at the same time. The data sources can be registered either as a System Data Source or as a User Data Source
- Lists all the DB2 System or User Data Sources that are already registered in the Microsoft Data Source Administrator
- Remove the System or User Data Sources that are already registered in the Microsoft Data Source Administrator

Note: The **db2cli registerdsn** is supported only on Microsoft Windows platforms.

Using security plugins with the IBM Data Server Driver for ODBC and CLI

A security plug-in is a dynamically-loadable library that provides authentication security services.

Procedure

Using security plug-ins with the IBM Data Server Driver for ODBC and CLI is no different from using security plug-ins with an IBM Data Server Client or IBM Data Server Runtime Client.

When you read about using security plug-ins throughout the DB2 Information Center and DB2 books, consider the IBM Data Server Driver for ODBC and CLI like an IBM Data Server Client. Details about using security plug-ins with IBM Data Server Clients apply to using security plug-ins with the IBM Data Server Driver for ODBC and CLI too.

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

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

Specification:

- CLI 2.1
- ODBC 1.0

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

Once a connection is established, the completed connection string is returned back in clear text. The completed returned string will contain any password in clear text for any password specified in the connection string, `db2cli.ini` file or

db2dsdriver.cfg configuration file. Applications can store this returned connection string for future connection requests.

Syntax

Generic

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

Function arguments

Table 4. *SQLDriverConnect* arguments

Data type	Argument	Use	Description
SQLHDBC	<i>ConnectionHandle</i>	input	Connection handle
SQLHWND	<i>WindowHandle</i>	input	Window handle. On Windows operating systems, this is the parent Windows handle. Currently the window handle is only supported on Windows. If a NULL is passed, then no dialog will be presented.
SQLCHAR *	<i>InConnectionString</i>	input	A full, partial or empty (null pointer) connection string (see following syntax and description).
SQLSMALLINT	<i>InConnectionStringLength</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>InConnectionString</i> .
SQLSMALLINT *	<i>OutConnectionString</i>	output	Pointer to buffer for the completed connection string. If the connection was established successfully, this buffer will contain the completed connection string. Applications should allocate at least SQL_MAX_OPTION_STRING_LENGTH bytes for this buffer.
SQLSMALLINT	<i>OutConnectionStringCapacity</i>	input	Number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function) needed to store <i>OutConnectionString</i> .
SQLSMALLINT *	<i>OutConnectionStringLengthPtr</i>	output	Pointer to the number of SQLCHAR elements (or SQLWCHAR elements for the Unicode variant of this function), excluding the null-termination character, available to return in the <i>OutConnectionString</i> buffer. If the value of <i>*OutConnectionStringLengthPtr</i> is greater than or equal to <i>OutConnectionStringCapacity</i> , the completed connection string in <i>OutConnectionString</i> is truncated to <i>OutConnectionStringCapacity</i> - 1 SQLCHAR or SQLWCHAR elements.

Table 4. *SQLDriverConnect* arguments (continued)

Data type	Argument	Use	Description
SQLUSMALLINT	<i>DriverCompletion</i>	input	Indicates when CLI should prompt the user for more information. Possible values: <ul style="list-style-type: none"> • SQL_DRIVER_PROMPT • SQL_DRIVER_COMPLETE • SQL_DRIVER_COMPLETE_REQUIRED • SQL_DRIVER_NOPROMPT

Usage

InConnectionString Argument

A request connection string has the following syntax:

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

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

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

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

where

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

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

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

OutConnectionString Argument

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

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

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

attribute-keyword ::= ODBC-attribute-keyword
| driver-defined-attribute-keyword

ODBC-attribute-keyword = {UID | PWD};[localized-identifier]

driver-defined-attribute-keyword ::= identifier[:localized-identifier]

attribute-value ::= {attribute-value-list} | ?

(The braces are literal; they are returned by CLI.)

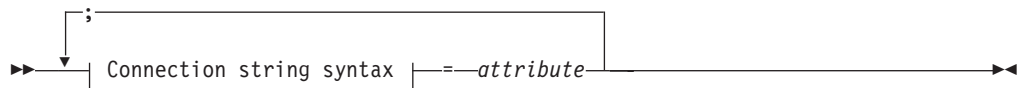
attribute-value-list ::= character-string [:localized-character
string] | character-string [:localized-character string], attribute-value-list

where

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

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

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



Connection string syntax



Attribute associated with each keyword are:

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

UID Authorization-name (user identifier).

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

NEWPWD

New password used as part of a change password request. The application can either specify the new string to use, for example, NEWPWD=newpass; or specify NEWPWD=; and rely on a dialog box

generated by the CLI driver to prompt for the new password (set the *DriverCompletion* argument to anything other than `SQL_DRIVER_NOPROMPT`).

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

If any keywords exists in the CLI initialization file, the keywords and their values are used to augment the information passed to CLI in the connection string. If the information in the CLI initialization file contradicts information in the connection string, the values in connection string take precedence.

If the end user *Cancels* a dialog box presented, `SQL_NO_DATA_FOUND` is returned.

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

SQL_DRIVER_PROMPT:

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

SQL_DRIVER_COMPLETE:

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

SQL_DRIVER_COMPLETE_REQUIRED:

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

SQL_DRIVER_NOPROMPT:

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

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

Unicode equivalent: This function can also be used with the Unicode character set. The corresponding Unicode function is `SQLDriverConnectW()`. See “Unicode functions (CLI)” on page 162 for information about ANSI to Unicode function mappings.

Return codes

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

Diagnostics

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

Table 5. *SQLDriverConnect SQLSTATES*

SQLSTATE	Description	Explanation
01004	Data truncated.	The buffer <code>szConnstrOut</code> was not large enough to hold the entire connection string. The argument <code>*OutConnectionStringLengthPtr</code> contains the actual length of the connection string available for return. (Function returns <code>SQL_SUCCESS_WITH_INFO</code>)
01S00	Invalid connection string attribute.	An invalid keyword or attribute value was specified in the input connection string, but the connection to the data source was successful anyway because one of the listed event has occurred: <ul style="list-style-type: none">• The unrecognized keyword was ignored.• The invalid attribute value was ignored, the default value was used instead. (Function returns <code>SQL_SUCCESS_WITH_INFO</code>)
HY000	General error. Dialog Failed	The information specified in the connection string was insufficient for making a connect request, but the dialog was prohibited by setting <code>fCompletion</code> to <code>SQL_DRIVER_NOPROMPT</code> . The attempt to display the dialog failed.
HY090	Invalid string or buffer length.	The value specified for <code>InConnectionStringLength</code> was less than 0, but not equal to <code>SQL_NTS</code> . The value specified for <code>OutConnectionStringCapacity</code> was less than 0.
HY110	Invalid driver completion.	The value specified for the argument <code>fCompletion</code> was not equal to one of the valid values.

Restrictions

None.

Example

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

CLI/ODBC keywords for file DSN or DSN-less connections

AltHostName CLI/ODBC configuration keyword:

Specifies the alternate host name to be used if the primary server specified by `HOSTNAME` cannot be contacted (Client Reroute.)

db2cli.ini keyword syntax:

`AltHostName = fully qualified alternate host name | IP address of node`

Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies a fully qualified host name or the IP address of the node where the alternate server for the database resides.

If the primary server returns alternate server information, it will override this `AltHostName` setting. However, this keyword is read only. That means the `db2cli.ini` will not be updated with the alternate server information received from the primary server.

AltPort CLI/ODBC configuration keyword:

Specifies the alternate port to be used if the primary server specified by `HOSTNAME` and `PORT` cannot be contacted (Client Reroute.)

db2cli.ini keyword syntax:

`AltPort = port number`

Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

This parameter specifies the port number of the alternate server of the database manager instance where the alternate server for the database resides.

If the primary server returns alternate server information, it will override this `AltPort` setting. However, this keyword is read only. That means the `db2cli.ini` will not be updated with the alternate server information received from the primary server.

Authentication CLI/ODBC configuration keyword:

Specifies the type of authentication to be used with file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

`Authentication = CERTIFICATE | SERVER | SERVER_ENCRYPT |
SERVER_ENCRYPT_AES | DATA_ENCRYPT | KERBEROS |
GSSPLUGIN`

Default setting:

Not specified

Usage notes:

This can be set in the [Data Source] section of the `db2cli.ini` file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- **Database**
- **Protocol.**

If `Protocol=IPC`, you need to set the following option as well:

- **Instance.**

If `Protocol=TCPIP`, you need to set the following options as well:

- **Port**
- **Hostname.**

If Kerberos is specified, then the **KRBPlugin** may also be optionally specified. If **KRBPlugin** is not specified, the default plug-in IBMkrb5 will be used.

Starting in DB2 Version 9.7 Fix Pack 6 and later, CERTIFICATE authentication is available for connection to DB2 for z/OS Version 10 with APAR PM53450 and later. The CERTIFICATE authentication type is supported starting in DB2 Version 9.7 Fix Pack 6. This authentication type allows you to use SSL client authentication without the need of providing database passwords on the database client. When certificate-based authentication is configured to supply authentication information, a password cannot be specified in any other way (as in the db2dsdriver.cfg configuration file, in the db2cli.ini configuration file, or in the connection string). If CERTIFICATE is specified, then the new label parameter SSLClientLabel must also be specified in the CLI configuration file, db2cli.ini, or in the data server driver configuration file, db2dsdriver.cfg.

BIDI CLI/ODBC configuration keyword:

Specifies the BIDI code page when connected to a DB2 for z/OS.

db2cli.ini keyword syntax:

BIDI = *code page*

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this option, you must also set the listed options:

- Database
- Protocol=TCPIP
- Hostname
- Port

FileDSN CLI/ODBC configuration keyword:

Specifies a DSN file from which a connection string will be built for the data source.

db2cli.ini keyword syntax:

You can not set this keyword in the db2cli.ini file.

You can specify the value of this keyword in the connection string in SQLDriverConnect like this:

FileDSN = *file name*

Instance CLI/ODBC configuration keyword:

Specifies the instance name for a local IPC connection for file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

Instance = *instance name*

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this keyword, you must also set the following options:

- Database
- Protocol=IPC

Interrupt CLI/ODBC configuration keyword:

Sets the interrupt processing mode.

db2cli.ini keyword syntax:

Interrupt = 0 | 1 | 2

Default setting:

1

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

When you set this option, you must also set the following options:

- Database
- Protocol=IPC

The keyword values have the following meaning:

- 0 Disables interrupt processing (SQLCancel calls will not interrupt the processing.)
- 1 Interrupts are supported (default.) In this mode, if the server supports an interrupt, an interrupt will be sent. Otherwise the connection is dropped.

The settings for INTERRUPT_ENABLED (a DB2 Connect™ gateway setting) and the DB2 registry variable DB2CONNECT_DISCONNECT_ON_INTERRUPT will take precedence over the Interrupt keyword setting of 1.
- 2 Interrupt drops the connection regardless of server's interrupt capabilities (SQLCancel will drop the connection.)

KRBPlugin CLI/ODBC configuration keyword:

Specifies the name of the Kerberos plug-in library to be used for client side authentication for file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

KRBPlugin = *plugin name*

Default setting:

By default, the value is null on UNIX operating systems, and IBMkrb5 on Windows operating systems.

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies the name of the Kerberos plug-in library to be used for client-side connection authentication. The plug-in is used when the client is authenticated using KERBEROS authentication.

Protocol CLI/ODBC configuration keyword:

Communications protocol used for File DSN or in a DSN-less connection.

db2cli.ini keyword syntax:

Protocol = **TCPIP** | **TCPIP6** | **TCPIP4** | **IPC** | **LOCAL**

Default setting:

none

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

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; and
- Hostname.

IPC connectivity can be specified by setting Protocol to either **IPC** or **LOCAL**.

When Protocol = **IPC** | **LOCAL** the Instance keyword must also be set.

PWDPlugin CLI/ODBC configuration keyword:

Specifies the name of the userid-password plug-in library to be used for client side authentication for file DSN or DSN-less connectivity.

db2cli.ini keyword syntax:

PWDPlugin = *plug-in name*

Default setting:

By default, the value is null and the DB2 supplied userid-password plug-in library is used.

Usage notes:

This can be set in the [Data Source] section of the db2cli.ini file for the given data source, or in a connection string.

This parameter specifies the name of the userid-password plug-in library to be used for client-side connection authentication. The plug-in is used when the client is authenticated using **SERVER** or **SERVER_ENCRYPT** authentication.

SaveFile CLI/ODBC configuration keyword:

Specifies the file name of a DSN file in which to save the attribute values of the keywords used in making the present, successful connection.

db2cli.ini keyword syntax:

You can not set this keyword in the db2cli.ini file.

You can specify the value of this keyword in the connection string in SQLDriverConnect like this:

SaveFile = *file name*

Running CLI and ODBC applications using the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured

separately; and it provides a subset of the functionality of either IBM Data Server Client. The IBM Data Server Driver for ODBC and CLI provides runtime support for: the CLI application programming interface (API), the ODBC API, the XA API; and connecting to databases.

Before you begin

To run database applications with the IBM Data Server Driver for ODBC and CLI, you need:

- one or more copies of the driver installed.
 - See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.
- to have configured the application environment for the driver.
 - See: “Configuring the IBM Data Server Driver for ODBC and CLI” on page 11.

Procedure

When writing applications for, or migrating applications to using the IBM Data Server Driver for ODBC and CLI:

- Ensure your applications use only the CLI, ODBC and XA API functions that are supported by the driver.
 - See:
 - “CLI and ODBC API support in the IBM Data Server Driver for ODBC and CLI”
 - “XA API support in the IBM Data Server Driver for ODBC and CLI” on page 36
- Ensure your applications are not attempting to make use of IBM Data Server Client or IBM Data Server Runtime Client functionality that is restricted in the driver.
 - See: “Restrictions of the IBM Data Server Driver for ODBC and CLI” on page 37
- - Use the 32-bit version of the driver with 32-bit database applications, and use the 64-bit version of the driver with 64-bit database applications.
- Understand the available tracing, logging, and diagnostic support provided by the driver for investigating problems.
 - See: “Diagnostic support in the IBM Data Server Driver for ODBC and CLI” on page 38

CLI and ODBC API support in the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server Client.

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server Client.

The IBM Data Server Driver for ODBC and CLI supports the ANSI and (where they exist) the Unicode versions of the following ODBC and CLI functions:

SQLAllocConnect	SQLExtendedPrepare	SQLNumParams
SQLAllocEnv	SQLFetch	SQLNumResultCols
SQLAllocHandle	SQLFetchScroll	SQLParamData
SQLAllocStmnt	SQLForeignKeys	SQLParamOptions
SQLBindCol	SQLFreeConnect	SQLPrepare
SQLBindFileToCol	SQLFreeEnv	SQLPrimaryKeys
SQLBindFileToParam	SQLFreeHandle	SQLProcedureColumns
SQLBindParameter	SQLFreeStmnt	SQLProcedures
SQLBrowseConnect	SQLGetConnectAttr	SQLPutData
SQLBuildDataLink	SQLGetConnectOption	SQLRowCount
SQLBulkOperations	SQLGetCursorName	SQLSetColAttributes
SQLCancel	SQLGetData	SQLSetConnectAttr
SQLCloseCursor	SQLGetDataLinkAttr	SQLSetConnectOption
SQLColAttribute	SQLGetDescField	SQLSetConnection
SQLColAttributes	SQLGetDescRec	SQLSetCursorName
SQLColumnPrivileges	SQLGetDiagField	SQLSetDescField
SQLColumns	SQLGetDiagRec	SQLSetDescRec
SQLConnect	SQLGetEnvAttr	SQLSetEnvAttr
SQLCopyDesc	SQLGetFunctions	SQLSetParam
SQLGetInfo	SQLSetPos	SQLDescribeCol
SQLGetLength	SQLSetScrollOptions	SQLDescribeParam
SQLGetPosition	SQLSetStmntAttr	SQLDisconnect
SQLGetSQLCA	SQLSetStmntOption	SQLDriverConnect
SQLGetStmntAttr	SQLSpecialColumns	SQLEndTran
SQLGetStmntOption	SQLStatistics	SQLError
SQLGetSubString	SQLTablePrivileges	SQLExecDirect
SQLGetTypeInfo	SQLTables	SQLExecute
SQLMoreResults	SQLTransact	SQLExtendedBind
SQLNativeSql	SQLExtendedFetch	SQLNextResult

XA API support in the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server client.

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server client.

The IBM Data Server Driver for ODBC and CLI supports the following XA API functions:

```

xa_open
xa_close
xa_start
xa_end
xa_prepare
xa_commit
xa_rollback
xa_forget
xa_recover

```

LDAP support in the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server client.

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server client.

The IBM Data Server Driver for ODBC and CLI supports the LDAP Database Directory support with one restriction:

- the LDAP cache is not saved to disk; it is an in-memory cache only. The DB2LDAPCACHE registry variable is ignored.

The steps for configuring the database application environment to enable LDAP when using the IBM Data Server Driver for ODBC and CLI are the same as for other scenarios, except that the DB2LDAPCACHE registry variable is ignored.

Restrictions of the IBM Data Server Driver for ODBC and CLI

IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server client.

IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM Data Server client.

IBM Data Server Driver for ODBC and CLI provides runtime support for:

- the DB2 CLI application programming interface (API)
- the ODBC API
- the XA API
- database connectivity
- the DB2 Interactive Call Level Interface (db2cli)

The following restrictions apply to the IBM Data Server Driver for ODBC and CLI:

- No other database product can be installed in the same path if the IBM Data Server Driver for ODBC and CLI is already installed.
- On Windows operating systems, you can install a maximum of 16 copies of the IBM Data Server Driver for ODBC and CLI.
- To connect to a z/OS server or a System i[®] server, you must register a DB2 Connect license key. (Retrieve the license file from your Passport Advantage[®] distribution, for example db2conpe.lic, then copy the license file to the license directory under the directory where the driver was installed.)
- XA connections against a z/OS server are supported. However, XA connections against a System i server are not supported.
- If you use the configuration file db2dsdriver.cfg to specify aliases, the following entries must contain a value:
 - <dsncollection> entries (alias, name, host, and port)
 - <database> entries (name, host, port).

These entries must be specified and cannot be empty.

- The CLI/ODBC configuration keyword DBNAME is not supported.
- The CLI LOAD utility statement attribute, sql_attr_use_load_api, is not supported.

Functionality not supported by the IBM Data Server Driver for ODBC and CLI

- CLI and ODBC application development
- the DB2 Command Line Processor (CLP)
- administrative APIs
- the CLIENT authentication type is not supported by the IBM Data Server Driver for ODBC and CLI and by the IBM Data Server Driver Package
- installation program
 - You must install the driver manually.
 - See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.
 - You must configure the driver manually.
 - See: “Configuring the IBM Data Server Driver for ODBC and CLI” on page 11.

Functionality supported with restrictions by the IBM Data Server Driver for ODBC and CLI

- Messages will be reported only in English.
- There is no local database directory.
 - LDAP is supported, but the LDAP cache is not saved to disk.
 - See: “LDAP support in the IBM Data Server Driver for ODBC and CLI” on page 36.
- Not all diagnostic utilities are available.
 - See: “Diagnostic support in the IBM Data Server Driver for ODBC and CLI.”

For an up-to-date list of current restrictions, see <http://www.ibm.com/developerworks/wikis/display/DB2/IBM+Data+Server+Driver+Limitations>.

Diagnostic support in the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM data server client.

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately; and it provides a subset of the functionality of either IBM data server client.

The following tracing, logging, and diagnostic utilities are provided with the IBM Data Server Driver for ODBC and CLI:

CLI trace

The method for using CLI trace with the IBM Data Server Driver for ODBC and CLI is the same as the method for using CLI trace with an IBM data server client.

DB2 trace

To turn on DB2 trace when you are using the IBM Data Server Driver for ODBC and CLI, you must invoke the db2trc utility from the adm subdirectory of the driver's install directory on Linux and UNIX or the bin subdirectory for Windows.

For example, if you installed the driver at \$HOME/ibm_data_server_driver_for_odbc_cli, you must be in the \$HOME/ibm_data_server_driver_for_odbc_cli/adm directory when you invoke db2trc.

The IBM Data Server Driver for ODBC and CLI can be installed on a Network File System (NFS). If the driver is installed on a read-only NFS, the environment variable DB2_DIAGPATH must be set to allow DB2 trace to work.

db2diag log files

When you are using the IBM Data Server Driver for ODBC and CLI, the **db2diag** log files will be located in the following locations:

- On Windows operating systems: The **db2diag** log file is located in the %SYSTEM APP DATA PATH%\IBM\DB2\%UNZIPPED PATH%. For example, if the CLI driver %UNZIPPED PATH% is D:\Program Files\IBM\clidriver then the db2diag.log path for the CLI driver will be %SYSTEM APP DATA PATH%\IBM\DB2\D_Program Files_IBM_clidriver, where %SYSTEM APP DATA PATH% is Drive:\Documents and Settings\All Users\Application Data\ for Windows 2003, Windows XP and Drive:\ProgramData\ for Windows 2008 and Windows Vista.
- On UNIX and Linux operating systems: The **db2diag** log file is located in the db2dump subdirectory of the driver install directory.

The DB2_DIAG environment variable and the DIAGPATH CLI keyword can be used to alter the location of the **db2diag** log files.

db2support

The DB2 command line processor is not available with the IBM Data Server Driver for ODBC and CLI, so the CLP utility is not available. However, an executable version of db2support will be available with the driver.

This executable version of db2support will collect the following information:

- db2level output;
- environment variables; and
- a listing of the contents of the IBM Data Server Driver for ODBC and CLI install directory.

You must invoke db2support from the bin subdirectory of the driver install directory.

For example, if you installed the driver at \$HOME/ibm_data_server_driver_for_odbc_cli, you must be in the \$HOME/ibm_data_server_driver_for_odbc_cli/bin directory when you invoke db2support.

Setting diagnostic options

The IBM Data Server Driver for ODBC and CLI does not support the command line processor (CLP.) This means that the usual mechanism to set DB2 registry variables, using the db2set command, is not possible. However, the functionality of registry variables related to diagnostics is supported through CLI/ODBC keywords:

- “DiagLevel CLI/ODBC configuration keyword” on page 40
- “NotifyLevel CLI/ODBC configuration keyword” on page 40

- “DiagPath CLI/ODBC configuration keyword” on page 41

and attributes for SQLSetEnvAttr and SQLGetEnvAttr:

- SQL_ATTR_DIAGLEVEL
- SQL_ATTR_NOTIFYLEVEL
- SQL_ATTR_DIAGPATH
- See: “Environment attributes (CLI) list” on page 41.

and environment variables:

- DB2_DIAGPATH
- See: “Environment variables supported by the IBM Data Server Driver for ODBC and CLI” on page 16.

The CLI/ODBC keyword DiagPath, the attribute SQL_ATTR_DIAGPATH, and the environment variable DB2_DIAGPATH all have the same purpose: to specify where diagnostic output is located. There is one situation, however, where DB2_DIAGPATH must be used:

- The IBM Data Server Driver for ODBC and CLI can be installed on a Network File System (NFS). If the driver is installed on a read-only NFS, the environment variable DB2_DIAGPATH must be set to a writable directory to allow DB2 trace to work.

Otherwise, the CLI/ODBC keyword DiagPath, the attribute SQL_ATTR_DIAGPATH, and the environment variable DB2_DIAGPATH all have the same effect.

DiagLevel CLI/ODBC configuration keyword:

Sets the diagnostic level.

db2cli.ini keyword syntax:

DiagLevel = 0 | 1 | 2 | 3 | 4

Default setting:

3

Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is applicable only at Environment Handle allocation time for an entire process.

This is equivalent to the database manager parameter DIAGLEVEL.

NotifyLevel CLI/ODBC configuration keyword:

Sets the diagnostic level.

db2cli.ini keyword syntax:

NotifyLevel = 0 | 1 | 2 | 3 | 4

Default setting:

3

Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is equivalent to the database manager parameter NOTIFYLEVEL.

DiagPath CLI/ODBC configuration keyword:

Sets the path of the **db2diag** log files.

db2cli.ini keyword syntax:

DiagPath = *existing directory*

Default setting:

The default value is the db2dump directory on UNIX and Linux operating systems, and the db2 directory on Windows operating systems.

Usage notes:

This can be set in the [COMMON] section of the db2cli.ini file only.

This is equivalent to the database manager parameter DIAGPATH.

Environment attributes (CLI) list:

CLI environment attributes that you can set using the SQLSetEnvAttr() function.

ODBC does not support setting driver-specific environment attributes using SQLSetEnvAttr(). Only CLI applications can set the CLI-specific environment attributes using this function.

SQL_ATTR_CONNECTION_POOLING

This attribute was deprecated in DB2 UDB for Linux, UNIX, and Windows Version 8.

This attribute is not supported when accessing the Informix database server.

SQL_ATTR_CONNECTTYPE

This attribute replaces the SQL_CONNECTTYPE attribute. A 32-bit integer value that specifies whether this application is to operate in a coordinated or uncoordinated distributed environment. The possible values are:

- **SQL_CONCURRENT_TRANS:** The application can have concurrent multiple connections to any one database or to multiple databases. Each connection has its own commit scope. No effort is made to enforce the coordination of the transaction. If an application issues a commit by using the environment handle on SQLEndTran() and not all of the connections commit successfully, the application is responsible for recovery. This is the default.
- **SQL_COORDINATED_TRANS:** The application can coordinate commit and rollbacks among multiple database connections. This option setting corresponds to the specification of the Type 2 CONNECT in embedded SQL. In contrast to the SQL_CONCURRENT_TRANS setting that was previously described, the application is permitted only one open connection per database.

Note: This connection type results in the default for the SQL_ATTR_AUTOCOMMIT connection option to be SQL_AUTOCOMMIT_OFF.

If you change this attribute from the default, you must set it before any connections are established on the environment handle.

Application typically set this attribute as an environment attribute with a call to SQLSetEnvAttr() function. The SQLSetEnvAttr() function is called as soon as the environment handle is allocated. However, because ODBC applications cannot access SQLSetEnvAttr() function, ODBC applications

must set this attribute using `SQLSetConnectAttr()` function after each connection handle is allocated, but before any connections are established.

All connections on an environment handle must have the same `SQL_ATTR_CONNECTTYPE` setting. An environment cannot have both concurrent and coordinated connections. The type of the first connection determines the type of all subsequent connections. `SQLSetEnvAttr()` returns an error if an application attempts to change the connection type while there is an active connection.

You can also set the default connect type by using the “ConnectType CLI/ODBC configuration keyword” on page 165.

The `SQL_ATTR_CONNECTTYPE` attribute is an IBM defined extension.

SQL_ATTR_CP_MATCH

This attribute was deprecated in DB2 database version 8.

This attribute is not supported when accessing the Informix database server.

SQL_ATTR_DB2TRC_STARTUP_SIZE

Description

A 32-bit integer value that allocates the DB2 trace facility buffer in MB.

Values

The `SQL_ATTR_DB2TRC_STARTUP_SIZE` value must be in the range of 1-1024 MB and in power of 2. If the `SQL_ATTR_DB2TRC_STARTUP_SIZE` value is not set to a value in power of 2, the specified `SQL_ATTR_DB2TRC_STARTUP_SIZE` value is rounded down to the closest power of 2 value.

Usage notes

The `SQL_ATTR_DB2TRC_STARTUP_SIZE` value only takes effect if the following conditions are met:

- No environment handle is allocated by the process that uses DB2 libraries.
- A trace buffer is not allocated before setting the value.

Once set, the trace facility buffer allocated by the `SQL_ATTR_DB2TRC_STARTUP_SIZE` cannot be changed while DB2 libraries are still loaded in the operating system.

The trace facility buffer is de-allocated when all running processes that use DB2 libraries exits.

If the trace facility buffer is already allocated, setting the `SQL_ATTR_DB2TRC_STARTUP_SIZE` attribute has no effect. The trace facility buffer can be allocated by:

- Issuing the **db2start** command on the IBM Data Server product.
- Issuing the **db2trc on** or **db2trc alloc** command.
- Setting the **db2trcStartupSize** keyword in the `db2dsdriver.cfg` file.
- Setting the `SQL_ATTR_DB2TRC_STARTUP_SIZE` environment attribute.

SQL_ATTR_DIAGLEVEL

Description

A 32-bit integer value which represents the diagnostic level. This is equivalent to the database manager DIAGLEVEL parameter.

Values

Valid values are: 0, 1, 2, 3, or 4. (The default value is 3.)

Usage notes

You must set this attribute before any connection handles are created.

SQL_ATTR_DIAGPATH**Description**

A pointer to a null-terminated character string that contains the name of the directory where diagnostic data is to be placed. The SQL_ATTR_DIAGPATH is equivalent to the database manager DIAGPATH parameter.

Values

The default value is the db2dump directory on UNIX and Linux operating systems, and the db2 directory on Windows operating systems.

Usage notes

You must set this attribute before any connection handles are created.

SQL_ATTR_INFO_ACCTSTR**Description**

A pointer to a null-terminated character string that is used to identify the client accounting string that is sent to the data server when DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 200 characters. A CLI application can set the SQL_ATTR_INFO_ACCTSTR attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (_) or period (.).

The SQL_ATTR_INFO_ACCTSTR attribute is an IBM-defined extension.

SQL_ATTR_INFO_APPLNAME**Description**

A pointer to a null-terminated character string that is used to identify the client application name that is sent to the data server when DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 32 characters. A CLI application can set the SQL_ATTR_INFO_APPLNAME attribute on DB2 for i V6R1 and

later servers. DB2 for i servers support a length of up to 255 characters. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (_) or period (.).

The SQL_ATTR_INFO_APPLNAME attribute is an IBM defined extension.

If you change the client application name and the accounting string is set by the WLM_SET_CLIENT_INFO procedure, the accounting string stored on the server is updated with the value of the accounting string from the client information.

SQL_ATTR_INFO_USERID

Description

A pointer to a null-terminated character string that is used to identify the client user ID that is sent to the data server when DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 16 characters. A CLI application can set the SQL_ATTR_INFO_USERID attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters.

Do not confuse the client user ID with the authentication user ID. The client user ID is for identification purposes only and is not used for any authorization. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (_) or period (.).

The SQL_ATTR_INFO_USERID attribute is an IBM defined extension.

If you change the client user ID and the accounting string is set by the WLM_SET_CLIENT_INFO procedure, the accounting string stored on the server is updated with the value of the accounting string from the client information.

SQL_ATTR_INFO_WRKSTNNAME

Description

A pointer to a null-terminated character string that is used to identify the client workstation name that is sent to the data server when using DB2 Connect or DB2 for Linux, UNIX, and Windows is used.

Values

Some servers might not be able to handle the entire length of the value and might truncate the value. DB2 for z/OS and DB2 Universal Database for z/OS and OS/390 servers support up to 18 characters. A CLI application can set the SQL_ATTR_INFO_WRKSTNNAME attribute on DB2 for i V6R1 and later servers. DB2 for i servers support a length of up to 255 characters. To ensure that the data is converted correctly when transmitted to a host system, use only the characters A - Z and 0 - 9 and the underscore (_) or period (.).

The `SQL_ATTR_INFO_WRKSTNNAME` attribute is an IBM defined extension.

If the `SQL_ATTR_INFO_WRKSTNNAME` attribute is not specified, a default value that consists of the host name is used. The host name is obtained by calling the `gethostname()` function. If the host name is not configured or an error is encountered during the `gethostname()` function call, no value for the `SQL_ATTR_INFO_WRKSTNNAME` attribute is sent to the server.

SQL_ATTR_MAXCONN

This attribute was deprecated in DB2 Version 8.

This attribute is not supported when accessing the Informix database servers.

SQL_ATTR_NOTIFYLEVEL

Description

A 32-bit integer value that represents the notification level. This is equivalent to the database manager `NOTIFYLEVEL` parameter.

Values

Valid values are: 0, 1, 2, 3, or 4. (The default value is 3.)

Usage notes

You must set this attribute value before any connection handles are created.

This attribute is not supported when accessing the Informix database servers.

SQL_ATTR_ODBC_VERSION

Description

A 32-bit integer that determines whether certain functionality exhibits ODBC 2.x (CLI v2) behavior or ODBC 3.0 (CLI v5) behavior. ODBC applications must set this environment attribute before calling any function that has an `SQLHENV` argument, or the call will return `SQLSTATE HY010` (Function sequence error.).

Values

The listed values are used to set the value of this attribute:

- `SQL_OV_ODBC3`: Causes the listed ODBC 3.0 (CLI v5) behavior:
 - CLI returns and expects ODBC 3.0 (CLI v5) codes for date, time, and timestamp.
 - CLI returns ODBC 3.0 (CLI v5) `SQLSTATE` codes when `SQLError()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` functions are called.
 - The *CatalogName* argument in a call to `SQLTables()` function accepts a search pattern.
- `SQL_OV_ODBC2`: Causes the listed ODBC 2.x (CLI v2) behavior:
 - CLI returns and expects ODBC 2.x (CLI v2) codes for date, time, and timestamp.
 - CLI returns ODBC 2.0 (CLI v2) `SQLSTATE` codes when `SQLError()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` functions are called.
 - The *CatalogName* argument in a call to `SQLTables()` function does not accept a search pattern.
- `SQL_OV_ODBC3_80`: Causes the listed ODBC 3.0 (CLI v5) behavior:

- CLI returns and expects ODBC 3.x codes for date, time, and timestamp.
- CLI returns ODBC 3.x SQLSTATE codes when `SQLError()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` functions are called.
- The *CatalogName* argument in a call to `SQLTables()` function accepts a search pattern.

SQL_ATTR_OUTPUT_NTS

Description

A 32-bit integer value that controls the use of null-termination in output arguments.

Values

The possible values are:

- **SQL_TRUE**: CLI uses null termination to indicate the length of output character strings (default).
- **SQL_FALSE**: CLI does not use null termination in output character strings.

The CLI functions that are affected by this attribute are all of the functions that are called for the environment (and for any connections and statements that are allocated under the environment) that have character string parameters.

You can set this attribute only when there are no connection handles that are allocated under this environment.

SQL_ATTR_PROCESSCTL

Description

A 32-bit mask that sets process-level attributes, which affect all environments and connections for the process. You must set this attribute before the environment handle is allocated.

The call to `SQLSetEnvAttr()` must have the *EnvironmentHandle* argument set to `SQL_NULL_HANDLE`. The settings remain in effect for the duration of the process. Generally, use this attribute only for performance sensitive applications, where large numbers of CLI function calls are being made. Before setting any of these bits, ensure that the application, and any other libraries that the application calls, comply with the restrictions that are listed.

Values

You can combine the listed values to form a bit mask:

- **SQL_PROCESSCTL_NOTHREAD** - This bit indicates that the application does not use multiple threads, or if it does use multiple threads, guarantees that all DB2 calls are serialized by the application. If set, CLI does not make any system calls to serialize calls to CLI, and sets the DB2 context type to `SQL_CTX_ORIGINAL`.
- **SQL_PROCESSCTL_NOFORK** - This bit indicates that the application will never fork a child process. By default, CLI does not check to see if an application forks a child process. However, if the `CheckForFork` CLI/ODBC configuration keyword is set, CLI checks the current process ID for each function call for all applications that are connecting to the database for which the keyword is enabled. You can set this attribute so that CLI does not check for forked processes for that application.

The `SQL_ATTR_PROCESSCTL` attribute is an IBM defined extension.

`SQL_ATTR_RESET_CONNECTION`

Description

A 32-bit unsigned integer value that specifies whether the ODBC Driver Manager notifies the ODBC drivers that a connection has been placed in the connection pool on Windows operating systems. If the `SQL_ATTR_ODBC_VERSION` environment attribute is set to `SQL_OV_ODBC3_80`, the ODBC Driver Manager sets this attribute before placing a connection in the connection pool so that the driver can reset the other connection attributes to their default values.

Values

The only possible value is:

- `SQL_RESET_CONNECTION_YES` (default): The ODBC Driver Manager notifies the ODBC drivers that a connection has been placed in the connection pool.

Note: You should use `SQL_ATTR_RESET_CONNECTION` only for communication between the ODBC Driver Manager and an ODBC driver. You should not set this attribute from an application because all connection attributes will be reset to their default value. For example, any connection attribute values that you set by using the `SQLSetConnectAttr ()` function will be reset to CLI default values and your application could behave unexpectedly.

`SQL_ATTR_SYNC_POINT`

This attribute was deprecated in DB2 database version 8.

This attribute is not supported when accessing the Informix database servers.

`SQL_ATTR_TRACE`

Description

A pointer to a null-terminated character string that is used to turn on the CLI/ODBC trace facility.

Values

The string must include the CLI keywords **TRACE** and **TRACEPATHNAME**. For example:

```
"TRACE=1; TRACEPATHNAME=<dir>";
```

Usage notes

This attribute is not supported when accessing the Informix database servers.

`SQL_ATTR_TRACENOHEADER`

Description

A 32-bit integer value that specifies whether header information is included in the CLI trace file.

Values

The possible values are:

- **0** - Header information is included in the CLI trace file.
- **1** - No header information is included in the CLI trace file.

You can use the SQL_ATTR_TRACENOHEADER attribute with an SQL_NULL_HANDLE or with a valid environment handle.

SQL_ATTR_USE_2BYTES_OCTET_LENGTH

This attribute is deprecated in DB2 database version 8.

This attribute is not supported when accessing the Informix database servers.

SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA

Setting this attribute is equivalent to setting the connection attribute SQL_ATTR_DESCRIBE_OUTPUT_LEVEL to 0.

SQL_ATTR_USE_LIGHT_OUTPUT_SQLDA is deprecated and applications should now use the connection attribute SQL_ATTR_DESCRIBE_OUTPUT_LEVEL.

SQL_ATTR_USER_REGISTRY_NAME

Description

This attribute is used only when authenticating a user on a server that is using an identity mapping service.

Values

The SQL_ATTR_USER_REGISTRY_NAME attribute is set to a user defined string that names an identity mapping registry. The format of the name varies depending on the identity mapping service. By providing this attribute you tell the server that the user name that is provided can be found in this registry.

After setting this attribute, the value is used on subsequent attempts to establish a normal connection, establish a trusted connection, or switch the user ID on a trusted connection.

Usage notes

This attribute is not supported when accessing the Informix database servers.

SQL_CONNECTTYPE

This *Attribute* is replaced with SQL_ATTR_CONNECTTYPE.

SQL_MAXCONN

This *Attribute* is replaced with SQL_ATTR_MAXCONN.

SQL_SYNC_POINT

This *Attribute* is replaced with SQL_ATTR_SYNC_POINT.

This attribute is not supported when accessing the Informix database servers.

Deploying the IBM Data Server Driver for ODBC and CLI with database applications

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client . It must be installed and configured separately. You can simplify the deployment of your CLI and ODBC database applications by creating an install program.

You can deploy the IBM Data Server Driver for ODBC and CLI with your CLI and ODBC database applications by obtaining the compressed file that contains the driver and following the installation and configuration steps required for the driver into your install program.

Before you begin

To deploy the IBM Data Server Driver for ODBC and CLI with your applications you will need:

- a mechanism for deploying the applications, such as an install program (install program is available on Windows only);
- to obtain the compressed file that contains the driver; See: “Obtaining the IBM Data Server Driver for ODBC and CLI” on page 8
- a redistribution license. See: “License requirements for the IBM Data Server Driver for ODBC and CLI.”

Restrictions

Under the terms of the redistribution license, only some of the IBM Data Server Driver for ODBC and CLI files can be redistributed. Which files may be redistributed is listed in the file `redist.txt`. This file can be found in the compressed file that contains the driver, called `ibm_data_server_driver_for_odbc_cli.zip` on the Windows operating systems and `ibm_data_server_driver_for_odbc_cli.tar.Z` on all other platforms.

Procedure

To incorporate the IBM Data Server Driver for ODBC and CLI into your install program:

1. Copy the driver files into your install program. See the restrictions mentioned previously about which driver files can be redistributed.
2. Set the install program to install the driver on the target machine. See: “Installing the IBM Data Server Driver for ODBC and CLI” on page 8.
3. Set the install program to configure the environment on the target machine. See: “Configuring the IBM Data Server Driver for ODBC and CLI” on page 11.

License requirements for the IBM Data Server Driver for ODBC and CLI

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately.

The IBM Data Server Driver for ODBC and CLI is not part of the IBM Data Server Client or the IBM Data Server Runtime Client. It must be installed and configured separately.

You can download and install the IBM Data Server Driver for ODBC and CLI and use it with your ODBC and CLI applications without a special license.

The IBM Data Server Driver for ODBC and CLI can connect to the following properly licensed servers:

- DB2 for Linux, UNIX, and Windows
- DB2 Connect Server
- InfoSphere® Federation Server
- IBM Informix
- DB2 for z/OS
- IBM DB2 for IBM i
- DB2 Server for VM and VSE

The IBM Data Server Driver for ODBC and CLI can be used to connect to DB2 for z/OS, IBM DB2 for IBM i, and DB2 Server for VM and VSE servers only if:

- a connection is established through a properly licensed DB2 Connect server; or
- directly to the server if and only if a properly formatted authentic DB2 Connect license file is present. The license file is distributed as part of the DB2 Connect products. The only way to obtain the license key file is to purchase one of the following DB2 Connect products:
 - DB2 Connect Personal Edition
 - DB2 Connect Enterprise Edition
 - DB2 Connect Application Server Edition
 - DB2 Connect Unlimited Edition for System z[®]
 - DB2 Connect Unlimited Edition for System i

No other product provides the required license file or the license rights afforded by the presence of this file. Tampering with or unauthorized distribution of this file is a breach of the license agreement. The relevant license file can be located on the DB2 Connect activation image in the `/db2/license` directory. The license must be copied to the `license` subdirectory of the data server driver installation path, for example: *installation_path*/license. The name of the file varies with the product:

- DB2 Connect Personal Edition: `db2conpe.lic`
- DB2 Connect Application Server Edition: `db2consv_as.lic`
- DB2 Connect Enterprise Edition: `db2consv_ee.lic`
- DB2 Connect Unlimited Edition for System i: `db2consv_is.lic`
- DB2 Connect Unlimited Edition for System z: `db2consv_zs.lic`

Chapter 3. ODBC driver managers

DB2 Call Level Interface (CLI) supports a variety of ODBC driver managers in connections to DB2.

An ODBC driver manager is not supplied on UNIX platforms as part of the operating system. Using ODBC on UNIX systems requires a separate commercial or open source ODBC driver manager. Refer to the unixODBC website (<http://www.unixodbc.org>), and the README files within the unixODBC distribution package for more information.

unixODBC driver manager

The unixODBC Driver Manager is an open source ODBC driver manager supported for DB2 ODBC applications on all supported Linux and UNIX operating systems.

Support statement

If you experience problems with the combination of the unixODBC Driver Manager and the DB2 ODBC driver after they have been properly installed and configured, you can contact DB2 Support (http://www.ibm.com/software/data/db2/support/db2_9/) for assistance in diagnosing the problem. If the source of the problem lies with the unixODBC Driver Manager, then you can:

- Purchase a service contract for technical support from Easysoft, a commercial sponsor of unixODBC (<http://www.easysoft.com>).
- Participate in any open source support channels at <http://www.unixodbc.org>.

Setting up the unixODBC driver manager

To run ODBC applications on Linux or UNIX operating systems, you must configure the unixODBC Driver Manager.

Procedure

To set up the unixODBC Driver Manager for use with CLI and ODBC applications:

1. Download the latest unixODBC source code from <http://www.unixodbc.org>.
2. Untar the source files. For example:

```
gzip -d unixODBC-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar
```

3. For AIX only: Configure the C compiler to be thread-enabled:

```
export CC=xlc_r
export CCC=x1C_r
```

4. To compile a 64-bit version of the driver manager using the xlc_r compilers, set the environment variables **OBJECT_MODE** and **CFLAGS**:

```
export OBJECT_MODE=64
export CFLAGS=-q64 -DBUILD_REAL_64_BIT_MODE
```

5. Install the driver manager in either your home directory or the default `/usr/local` prefix:

- (Home directory) Issue the following command in the directory where you untarred the source files:

```
./configure --prefix=$HOME -DBUILD_REAL_64_BIT_MODE --enable-gui=no
--enable-drivers=no
```

- (/usr/local as root) Issue the following command:
./configure --enable-gui=no --enable-drivers=no
6. Optional: Examine all configuration options by issuing the following command:
./configure --help
 7. Build and install the driver manager:
make
make install

Libraries will be copied to the [prefix]/lib directory, and executables will be copied to the [prefix]/bin directory.

8. For AIX only: Extract the shared library from the ODBC driver for DB2 to yield shr.o on 32-bit operating systems and shr_64.o on 64-bit operating systems. To avoid confusion, rename the files db2.o and db2_64.o. These steps are necessary on AIX because the unixODBC Driver Manager loads the driver dynamically.
 - On 32-bit operating systems, issue the following commands:
cd *INSTHOME*/sql/lib/lib
ar -x libdb2.a
mv shr.o db2.o

where *INSTHOME* is the home directory of the instance owner.

- On 64-bit operating systems, issue the following commands:
cd *INSTHOME*/sql/lib/lib
ar -x -X 64 libdb2.a
mv shr_64.o db2_64.o

where *INSTHOME* is the home directory of the instance owner.

Ensure that your INI file references the correct library.

9. Optional: For AIX only: Extract libodbc.a, libodbcinst.a, and libodbccr.a if you will be dynamically loading the driver manager:
ar -x libodbc.a
ar -x libodbcinst.a
ar -x libodbccr.a

This produces libodbc.so.1, libodbcinst.so.1, and libodbccr.so.1 in the [prefix]/lib/so directory.

10. Build the application and ensure it is linked to the unixODBC Driver Manager by including the -L[prefix]/lib -lodbc option in the compile and link command.
11. Specify the paths for at least the user INI file (odbc.ini) or the system INI file (odbcinst.ini), and set the **ODBCHOME** environment variable to the directory where the system INI file was created.

Important: Provide absolute paths when specifying the paths of the user and system INI files. Do not use relative paths or environment variables.

Note: If compiling 64-bit applications for the ODBC Driver, use the -DODBC64 option to enable the 64-bit definitions in the driver manager.

Microsoft ODBC driver manager

The Microsoft ODBC driver manager can be used for connections to remote DB2 databases when using TCP/IP networks.

DataDirect ODBC driver manager

The DataDirect ODBC driver manager for DB2 can be used for connections to the DB2 database.

Restrictions

Complications arise when using the CLI/ODBC driver with the DataDirect Connect for ODBC Driver Manager in the UNIX environment because of the use of UTF-8 character encoding by the driver manager. UTF-8 is a variable length character encoding scheme using anywhere from 1 to 6 bytes to store characters. UTF-8 and UCS-2 are not inherently compatible, and passing UTF-8 data to the CLI/ODBC driver (which expects UCS-2) might result in application errors, data corruption, or application exceptions.

To avoid this problem, the DataDirect Connect for ODBC Driver Manager 4.2 Service Pack 2 recognizes a CLI/ODBC driver and not use the Unicode functions, effectively treating the CLI/ODBC driver as an ANSI-only driver. Before release 4.2 Service Pack 2, the DataDirect Connect for ODBC Driver Manager had to be linked with the _36 version of the CLI/ODBC driver which does not export the SQLConnectW function.

Chapter 4. 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), for all subsequent calls that require an environment handle.

2. Optional: Set environment attributes for your application by calling `SQLSetEnvAttr()` with the required 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 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), for all subsequent calls that require a connection handle.

4. Optional: Set connection attributes for your application by calling `SQLSetConnectAttr()` with the required 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);
```

Results

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

Initialization and termination in CLI overview

Conceptual view of initialization and termination tasks 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.

Figure 2 on page 57 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 Chapter 6, "Transaction processing in CLI overview," on page 73.

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 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.

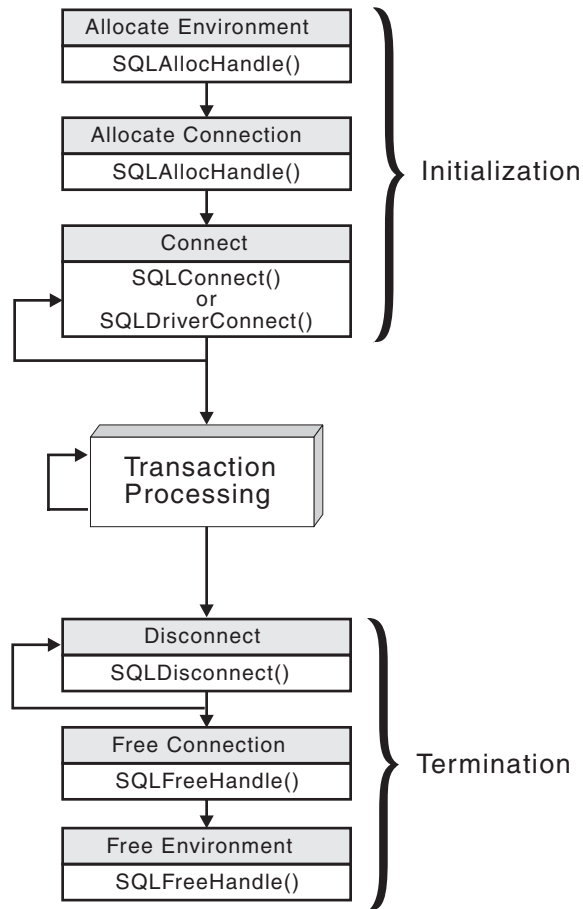


Figure 2. Conceptual view of initialization and termination tasks

Handles in CLI

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

A CLI handle is a variable that refers to a data object allocated and managed by 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 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, 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 $(3 * 64) + (30 * 384) = 11712$.

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. 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 CLI resources.

Chapter 5. Data types and data conversion in CLI applications

When writing a 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.

When writing a 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 CLI functions to transfer data between the DBMS and the application.

To facilitate this, 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. 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 CLI driver.

Note:

- GRAPHIC and VARGRAPHIC columns are not supported by Informix data server. Due to this limitation, conversion from `sql_c_dbchar` (C data type) and `sql_graphic` (SQL Datatype) are not supported. NCHAR and NVARCHAR datatypes and SQL_C_BINARY and SQL_BINARY conversions may be used instead of GRAPHIC and VARGRAPHIC.
- The SQL_XML data type is not supported for use with an Informix data server.

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 following code snippet 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

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.

Note: When using CLI with an Informix data server, binary values can not be inserted into CHAR columns. User can only insert binary values into a LOB column. This is an Informix data server limitation on converting to STRING types. When binding a TIMESTAMP column to a sql_c_char/sql_c_wchar string with an Informix data server, the input value must be specified using the ODBC escape sequence or as a literal. DATETIME functions can not be specified.

The rules that specify limits on precision and scale, as well as truncation and rounding rules for type conversions apply in 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 about the failure. When moving and converting floating point data values between the application and CLI, no correspondence is guaranteed to be exact as the values may change in precision and scale.

String handling in CLI applications

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

The following conventions deal with the various aspects of string arguments in 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, 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 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, CLI will not return a length. When the output data is a NULL value, 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, 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()`, CLI would update both values in one operation.

Null-termination of strings

By default, every character string that 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 CLI can indicate the actual length of data returned; otherwise, the application will not have any means to determine this. The CLI default is to always write the null terminator.

It is possible, using the Patch1 CLI/ODBC configuration keyword, to force CLI to null terminate graphic and DBCLOB strings.

String truncation

If an output string does not fit into a buffer, 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 `01004` 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 `01004`, 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, 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 quotation marks) the leading and trailing blanks are removed and case is ignored.

Blank padding of strings

DB2 Universal Database Version 8.1.4 and later do not pad strings with blanks to fit the column size, as was the behavior in releases of DB2 UDB from Version 8.1 through to Version 8.1.4. With DB2 UDB Version 8.1.4 and later, a string might have a length which differs from the length defined for the CHAR column if code page conversion occurred. For releases of DB2 UDB before Version 8.1.4, strings are padded with blanks to fill the column size; these blanks would be returned as part of the string data when the string was fetched from the CHAR column.

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).

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`. The LOB symbolic constants can be specified or returned on any of the CLI functions that take in or return an SQL data type argument (such as `SQLBindParameter()`, `SQLDescribeCol()`).

LOB locators versus file input and output

By default row data is returned with LOB locators. For example, if a CLI application does not provide an output buffer, the IBM data server client will request a LOB locator on behalf of the application for each LOB column in the

result set. However, if the application binds a buffer of adequate size to a LOB column, the LOB value will be returned in the buffer.

When a CLI application calls the function `SQLGetData()` to retrieve the LOB data, it will, by default, make one request to the server, and will store the entire LOB in memory provided *BufferLength* is large enough. If *BufferLength* is not large enough to hold the entire LOB value, it will be retrieved piecewise.

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.

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.

Note: When accessing Informix database servers, Large Binary Object blocking is not supported.

Figure 3 on page 64 shows the retrieval of a character LOB (CLOB).

- The left 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 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.

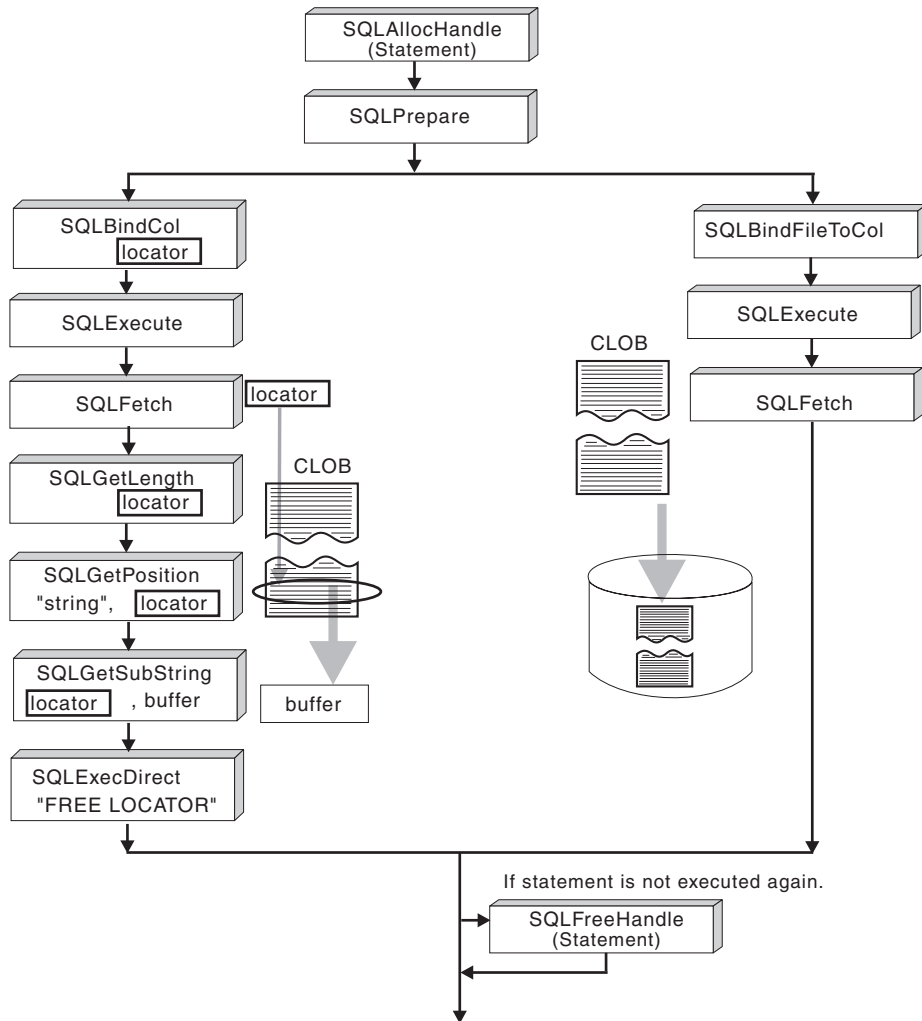


Figure 3. Fetching CLOB data

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).

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.

In a CLI application, for a statement that retrieves LOB data, by default the row data is returned with LOB locators to reference the LOB values. In cases where a buffer of an adequate size has been bound to a LOB column, the LOB value will be returned in the buffer and not as a LOB locator.

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()`).

Examples of using LOB locators

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. 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))
```

CLI applications may also obtain LOB values in pieces using the following VALUES statement:

```
VALUES (SUBSTR(:locator, :offset, :length))
```

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 CLI LOB file access functions are `SQLBindFileToCol()` and `SQLBindFileToParam()`.

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 CLI LOB file access functions are:

SQLBindFileToCol()

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

Example:

```
SQLINTEGER    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:

```
SQLINTEGER    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: 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.

LOB usage in ODBC applications

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, CLI will map the ODBC long data types to the DB2 LOB data types.

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, 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.

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()`.

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 as long as `SQLParamData()` returns `SQL_NEED_DATA`.
 - 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).

On DB2 for z/OS, calls to the `SQLEndTran()` function specifying `SQL_ROLLBACK` as completion type are allowed when the `SQL_ATTR_FORCE_ROLLBACK` connection attribute is set, the `StreamPutData` configuration keyword is set to 1, and autocommit mode is enabled.

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.

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.

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.

Note: User-defined types (UDTs) are not currently supported by CLI when running with an Informix database server. Using a UDT with an Informix database server will return CLI Error -999 [IBM][CLI Driver][IDS] Not implemented yet.

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.

Descriptor fields return a 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 want 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);
```

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.

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.

XML data handling in CLI applications - Overview

CLI applications can retrieve and store XML data by using the `SQL_XML` data type. This data type corresponds to the native XML data type of the DB2 database, which is used to define columns that store well-formed XML documents. You can bind the `SQL_XML` type to the following C types: `SQL_C_BINARY`, `SQL_C_CHAR`, `SQL_C_WCHAR`, and `SQL_C_DBCHAR`.

CLI applications can retrieve and store XML data by using the `SQL_XML` data type. This data type corresponds to the native XML data type of the DB2 database, which is used to define columns that store well-formed XML documents. You can bind the `SQL_XML` type to the following C types: `SQL_C_BINARY`, `SQL_C_CHAR`, `SQL_C_WCHAR`, and `SQL_C_DBCHAR`. Use the default `SQL_C_BINARY` type, instead of character types, to avoid possible data loss or corruption, which could result from code page conversion when character types are used.

To store XML data in an XML column, bind a binary (`SQL_C_BINARY`) or character (`SQL_C_CHAR`, `SQL_C_WCHAR`, or `SQL_C_DBCHAR`) buffer that contains the XML value to the `SQL_XML` SQL type and execute the `INSERT` or `UPDATE` SQL statements. To retrieve XML data from the database, bind the result set to a binary (`SQL_C_BINARY`) or character (`SQL_C_CHAR`, `SQL_C_WCHAR`, or `SQL_C_DBCHAR`) type. You should use character types with caution because of encoding issues.

When an XML value is retrieved into an application data buffer, the DB2 server performs an implicit serialization on the XML value to convert it from its stored hierarchical form to the serialized string form. For character-typed buffers, the XML value is implicitly serialized to the application character code page that is associated with the character type.

By default, an XML declaration is included in the output serialized string. You can change this default behavior by setting the `SQL_ATTR_XML_DECLARATION` statement or connection attribute, or by setting the `XMLDeclaration CLI/ODBC` configuration keyword in the `db2cli.ini` file.

You can issue and execute XQuery expressions and SQL/XML functions in CLI applications. SQL/XML functions are issued and executed like any other SQL statements. You must add a prefix to the XQuery expressions with the not case sensitive keyword **XQUERY**, or you must set the `SQL_ATTR_XQUERY_STATEMENT` statement attribute for the statement handle that is associated with the XQuery expression.

Note: Starting with DB2 Version 9.7 Fix Pack 5, the SQL_XML data type is supported for DB2 for i V7R1 servers or later releases.

Changing of default XML type handling in CLI applications

CLI supports CLI/ODBC configuration keywords that provide compatibility for applications that do not expect the default types returned when describing or specifying SQL_C_DEFAULT for XML columns and parameter markers.

CLI supports CLI/ODBC configuration keywords that provide compatibility for applications that do not expect the default types returned when describing or specifying SQL_C_DEFAULT for XML columns and parameter markers. Older CLI and ODBC applications might not recognize or expect the default SQL_XML type when describing XML columns or parameters. Some CLI or ODBC applications might also expect a default type other than SQL_C_BINARY for XML columns and parameter markers. To provide compatibility for these types of applications, CLI supports the MapXMLDescribe and MapXMLCDefault keywords.

MapXMLDescribe specifies which SQL data type is returned when XML columns or parameter markers are described.

MapXMLCDefault specifies the C type that is used when SQL_C_DEFAULT is specified for XML columns and parameter markers in CLI functions.

Chapter 6. Transaction processing in CLI overview

Transaction processing shows the typical order of function calls in the transaction processing task of a CLI application. Not all functions or possible paths are shown.

Figure 4 shows the typical order of function calls in the transaction processing task of a CLI application. Not all functions or possible paths are shown.

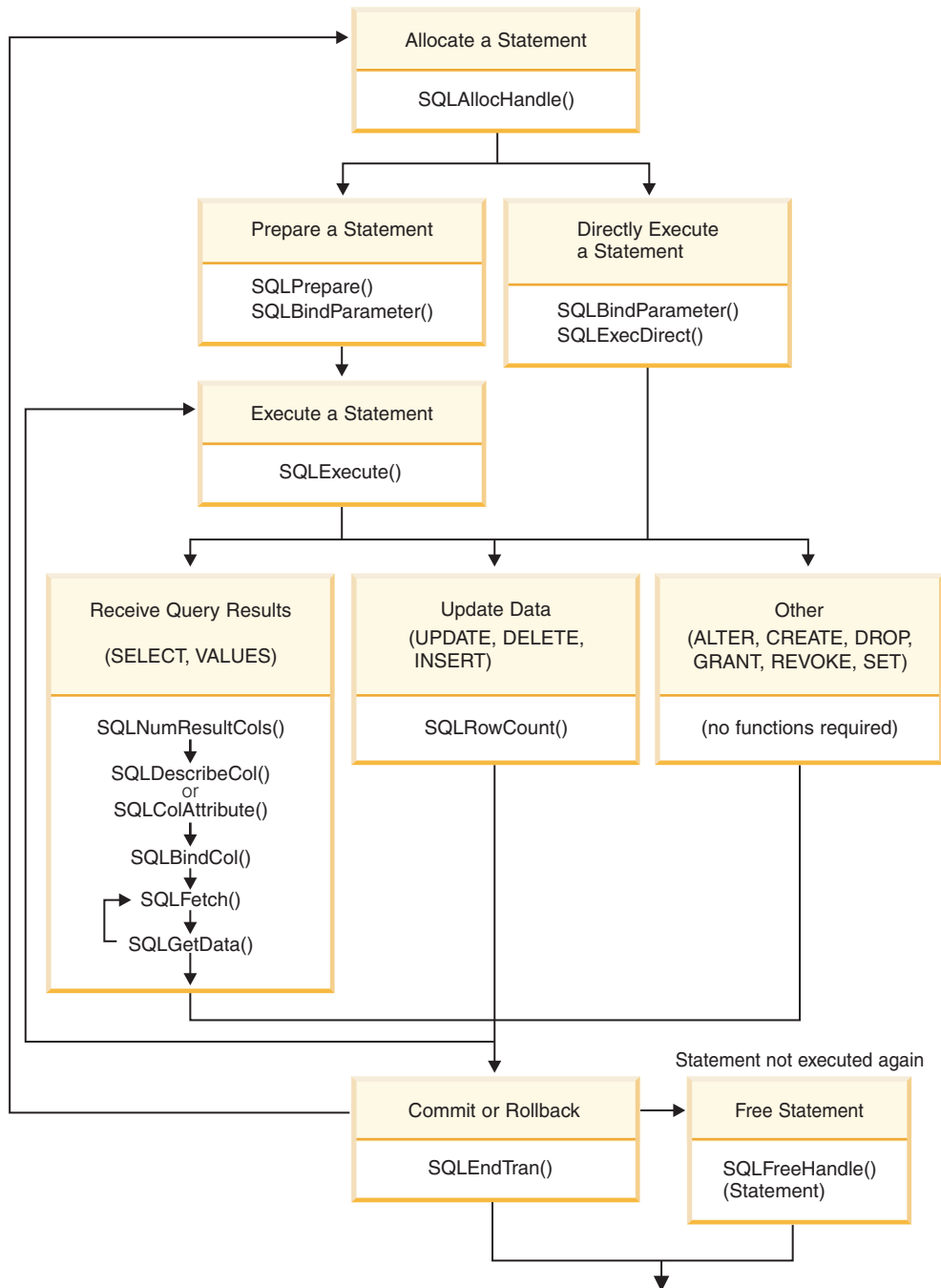


Figure 4. Transaction processing

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.

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.

Before you begin

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 required attribute option.

Results

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

Issuing SQL statements in CLI applications

SQL statements are passed to 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 CLI applications.

Before you begin

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:
 1. 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');";
```

2. Call SQLExecDirect() to issue the first statement in the statement handle and then call SQLMoreResults() to issue subsequent statements in the statement handle, as shown in the following example:

```
/* Issuing the first SELECT statement of multiple_stmts */
cliRC = SQLExecDirect (hstmt, multiple_stmts, SQL_NTS);
/* ... process result-set of first SELECT statement ... */

/* Issuing the second SELECT statement of multiple_stmts */
cliRC = SQLMoreResults(hstmt);
/* ... process result-set of second SELECT statement ... */

/* Issuing the INSERT statement of multiple_stmts */
cliRC = SQLMoreResults(hstmt);
/* cliRC is set to SQL_NO_DATA_FOUND to indicate that */
/* there are no more SQL statements to issue */
```

When a list of SQL statements is specified on the same statement handle, only one statement is issued at a time, starting with the first statement in the list. Each subsequent statement is issued in the order it appears.

- To issue SQL statements with parameter markers, see “Binding parameter markers in CLI applications” on page 77.
- To capture and convert SQL statements dynamically executed with CLI (dynamic SQL) to static SQL, see “Creating static SQL by using CLI/ODBC static profiling” on page 123.

Parameter marker binding in CLI applications

Parameter markers indicate the position in the SQL statement where the contents of application variables are to be substituted when the statement is executed.

Parameter markers 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 can be used in static embedded SQL.) CLI supports unnamed parameter markers, which are represented by a question mark (?), and named parameter markers, which are represented by a colon followed by a name (for example, *:name*, where *name* is a valid identifier). To

use named parameter markers, you must explicitly enable named parameter processing by setting the `EnableNamedParameterSupport` configuration keyword to `TRUE`.

Parameter markers can be bound to:

- 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, CLI will transfer the contents of the file directly to the database server.

An application cannot place parameter markers in the listed locations:

- In a `SELECT` list
- As both expressions in a comparison-predicate
- As both operands of a binary operator
- As both the first and second operands of a `BETWEEN` operation
- As both the first and third operands of a `BETWEEN` operation
- As both the expression and the first value of an `IN` operation
- As the operand of a unary `+` or `-` operation
- As the argument of a `SET FUNCTION` reference

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 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 CLI accesses them at execution time. Doing so can result in 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 CLI will convert the contents of the variable to match the SQL data type specified. For example, the SQL statement might require an integer value, but your application has a string representation of an integer. The string can be bound to the parameter, and CLI will convert the string to the corresponding integer value when you execute the statement.

By default, CLI does not verify the type of the parameter marker. If the application indicates an incorrect type for the parameter marker, it might cause:

- an extra conversion by the DBMS
- an error at the DBMS which forces 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 might want 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()` must be called with `SQL_RESET_PARAMS` to reset or clear the parameter bindings.

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.

Before you begin

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 inputbinding parameter markers with column-wise array input
 - binding parameter markers with row-wise array inputbinding parameter markers with row-wise 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.

Before you begin

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

About this task

For character and binary input data, the application uses the maximum input buffer size argument (*BufferLength*) of the `SQLBindParameter()` call to indicate to 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_PARAM_BIND_TYPE` statement attribute to `SQL_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.

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.

Before you begin

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 be 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,
                     3, 0, R[0].B, 3, &R.Lb);

/* Parameter C */
rc = SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR, SQL_CHAR,
                     10, 0, R[0].C, 10, &R.Lc);

```

Parameter diagnostic information in CLI applications

A *parameter status array* is an array of one or more `SQLSMALLINT`s allocated by a CLI application. Each element in the array corresponds to an element in the input (or output) parameter array.

A *parameter status array* is an array of one or more `SQLSMALLINT`s allocated by a CLI application. Each element in the array corresponds to an element in the input (or output) parameter array. If specified, the 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.

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 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()`.

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, 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()`.

Before you begin

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 value to the location 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 as required.

Specifying parameter values at execute time for long data manipulation in CLI applications

When manipulating long data, it might 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.

Before you begin

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

About this task

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

- SQLParamData() and SQLPutData() functions.
- The SQLCancel() function which is used to cancel the flow and force an exit from the loops without executing the SQL statement.
- The SQLGetDiagRec() function.

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.

Procedure

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.

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);
        /* ... */
    }
}

```

Results

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() might have produced.

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.

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.

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 CLI is in manual-commit mode, a new transaction will be implicitly started if the application is not already in a transaction and an SQL statement that can be contained within a transaction is executed.

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 because locks are removed as soon as possible. Applications that must 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 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.

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 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 behavior 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.

Preparing and executing SQL statements in CLI applications

After you have allocated a statement handle, you can perform operations using SQL statements or XQuery expressions.

An SQL statement or XQuery expression must be prepared before it can be executed, and CLI offers two ways of preparing and executing: perform the prepare and execute operations in separate steps, and combine the prepare and execute operations into one step.

Before you begin

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

Procedure

- To prepare and execute an SQL statement or XQuery expression in separate steps:
 1. Prepare the SQL statement or XQuery expression by calling `SQLPrepare()` and passing the statement or expression as the *StatementText* argument.

Note: XQuery expressions must be prefixed with the case-insensitive "XQUERY" keyword, unless the statement attribute `SQL_ATTR_XQUERY_STATEMENT` has been set to `SQL_TRUE` for this statement handle.
 2. Call `SQLBindParameter()` to bind any parameter markers you have in the SQL statement. CLI supports named parameter markers (for example, `:name`) and unnamed parameter markers represented by a question mark (?).

Note:

- To use named parameter markers, you must explicitly enable named parameter processing by setting the `EnableNamedParameterSupport` configuration keyword to `TRUE`.
 - For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the `XMLQUERY` function to bind parameter markers to XQuery variables. The values of the bound parameter markers are then passed to the XQuery expression specified in `XMLQUERY` for execution.
3. Execute the prepared statement by calling `SQLExecute()`.

Use this method when:

- The same SQL statement or XQuery expression is executed repeatedly (usually with different parameter values). This avoids having to prepare the same statement or expression more than once. The subsequent executions use the access plans already generated by the prepared statement, thus increasing driver efficiency and delivering better application performance.
 - The application requires information about the parameters or columns in the result set before the statement execution.
- To prepare and execute an SQL statement or XQuery expression in one step:
 1. Call `SQLBindParameter()` to bind any parameter markers you may have in the SQL statement. CLI supports named parameter markers (for example, `:name`) and unnamed parameter markers represented by a question mark (?).

Note:

- To use named parameter markers, you must explicitly enable named parameter processing by setting the `EnableNamedParameterSupport` configuration keyword to `TRUE`.
 - For XQuery expressions, you cannot specify parameter markers in the expression itself. You can, however, use the `XMLQUERY` function to bind parameter markers to XQuery variables. The values of the bound parameter markers are then passed to the XQuery expression specified in `XMLQUERY` for execution.
2. Prepare and execute the statement or expression by calling `SQLExecDirect()` with the SQL statement or XQuery expression as the *StatementText* argument.

Note: XQuery expressions must be prefixed with the case-insensitive "XQUERY" keyword, unless the statement attribute `SQL_ATTR_XQUERY_STATEMENT` has been set to `SQL_TRUE` for this statement handle.

3. Optional: If a list of SQL statements is 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 or expression is executed only once. This avoids having to call two functions to execute the statement or expression.
- The application does not require information about the columns in the result set before the statement is executed.

Deferred prepare in CLI applications

Deferred prepare is the name of the CLI feature that seeks to minimize communication with the server by sending both the prepare and execute requests for SQL statements in the same network flow.

Deferred prepare is the name of the CLI feature that seeks to minimize 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_DEFERRED_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 resource usage 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 before 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.

Executing compound SQL (CLI) statements in CLI applications

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.

About this task

- Compound SQL (CLI) does not guarantee the order in which the substatements are executed, therefore there must not be any dependencies among the substatements.
- Compound SQL (CLI) 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.
- Atomic array input is not supported within a BEGIN COMPOUND and END COMPOUND block of SQL statements. Atomic array input refers to the behavior where all inserts will be undone if any single insert fails.
- 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()
 - SQLParamData()
 - SQLPrepare()
 - SQLPutData()

Procedure

To execute compound SQL (CLI) statements in CLI applications:

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. If this is an atomic compound SQL operation, execute the substatements using the SQLExecute() function only. For example:

```
SQLExecute (hstmtsub1);
SQLExecute (hstmtsub2);
SQLExecute (hstmtsub3);
```

Note: All statements to be executed inside an atomic compound block must first be prepared. Attempts to use the SQLExecDirect() function within an atomic compound block will result in errors.
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);
```

Results

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.

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.

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. With the introduction of the DB2 Universal Database Version 8 client, the support for an updatable scrollable cursor was moved from the client to the server. This means that an application using the DB2 UDB client, Version 8 or higher, that requires updatable scrollable cursor needs to ensure that the server supports updatable scrollable cursors. DB2 UDB servers on Linux, UNIX and Windows, Version 8 and later, and DB2 for z/OS servers Version 7 and later support this feature. To access a scrollable cursor in a three-tier environment on DB2 for z/OS Version 7 and later, the gateway must be running DB2 UDB Version 8 or later.

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 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 CLI:

non-scrollable

Forward-only non-scrollable cursors are the default cursor type used by the CLI driver. This cursor type is unidirectional and requires the least amount of resource utilization.

scrollable

There are three types of scrollable cursors supported by CLI:

static This is a read-only cursor. When 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 might or might 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: CLI currently only supports keyset-driven cursors if the server supports them. The DB2 UDB Version 8 server now supports updatable scrollable cursors. This means that if applications require keyset cursor functionality and currently

access DB2 for OS/390 Version 6 or DB2 for UNIX and Windows Version 7 and earlier, the clients should not be upgraded to DB2 UDB Version 8 or later. The servers can be upgraded to Version 8 or later. Informix database servers do not support using keyset-driven cursors.

dynamic

Dynamic scrollable cursors can detect all changes (inserts, deletes, and updates) to the result set, and make insertions, deletions and updates to the result set. Unlike keyset-driven cursors, dynamic cursors:

- detect rows inserted by other cursors
- omit deleted rows from the result set (keyset-driven cursors recognize deleted rows as "holes" in the result set)

Currently, dynamic scrollable cursors are only supported in CLI when accessing servers that are DB2 for z/OS Version 8.1 and later. Dynamic scrollable cursors are not supported when accessing Informix database servers.

Note: A column with a LOB type, distinct type on a LOB type, A column with a LONG VARCHAR, LONG VARGRAPHIC, DATALINK, LOB, XML type, distinct type on any of these types, or structured type cannot be specified in the select-list of a scrollable cursor. CLI will downgrade the cursor type from scrollable to forward-only and return a CLI0005W (SQLSTATE 01S02) warning message.

Cursor attributes

The table 1 lists the default attributes for cursors in CLI.

Table 6. 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
dynamic ^b	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.
 • **b** Values concurrency is the default behavior, however, DB2 on Linux, UNIX and Windows will also support lock concurrency, which will result with pessimistic locking.

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 default 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. If lock concurrency has been explicitly asked for, then pessimistic locking will be used and locks will be held as soon as the row is read. This level of locking is only supported against DB2 on Linux, UNIX and Windows servers. 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 must 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 through 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. However, the inserted rows will appear to be part of the result set as 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.

Cursor considerations for CLI applications

The first decision to make is between a forward-only cursor and a scrollable cursor. A forward-only cursor incurs less resource usage than a scrollable cursor, and scrollable cursors have the potential for decreased concurrency.

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 resource usage 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 non-scrollable cursor.

If a scrollable cursor is required then you must decide between a static cursor, a keyset-driven cursor, or a dynamic cursor. A static cursor involves the least amount of resource usage. If the application does not need the additional features of a keyset-driven or dynamic cursor then a static cursor should be used.

Note: Currently, dynamic cursors are only supported when accessing servers that are DB2 for z/OS Version 8.1 and later.

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 either a keyset-driven or dynamic cursor. To perform updates and deletions on rows in a dynamic scrollable cursor's result set, the UPDATE or DELETE statement must include all the columns of at least one unique key in the base table. This can be the primary key or any other unique key. Because dynamic cursors incur more resource usage and might have less concurrency than keyset-driven cursors, only choose dynamic cursors if the application needs to detect both changes made and rows inserted by other cursors.

If an application requests a scrollable cursor that can detect changes without specifying a particular cursor type, then CLI will assume that a dynamic cursor is not needed and provide a keyset-driven cursor. This behavior avoids the increased resource usage and reduced concurrency that is incurred with dynamic cursors.

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 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 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.

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 Linux, UNIX and Windows might 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 7. Configuration keyword values restoring application behavior before scrollable cursor support

Configuration keyword setting	Description
Patch2=6	Returns a message that scrollable cursors (keyset-driven, dynamic 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 or dynamic cursor is requested.

Result set terminology in CLI applications

The following terms describe result handling.

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

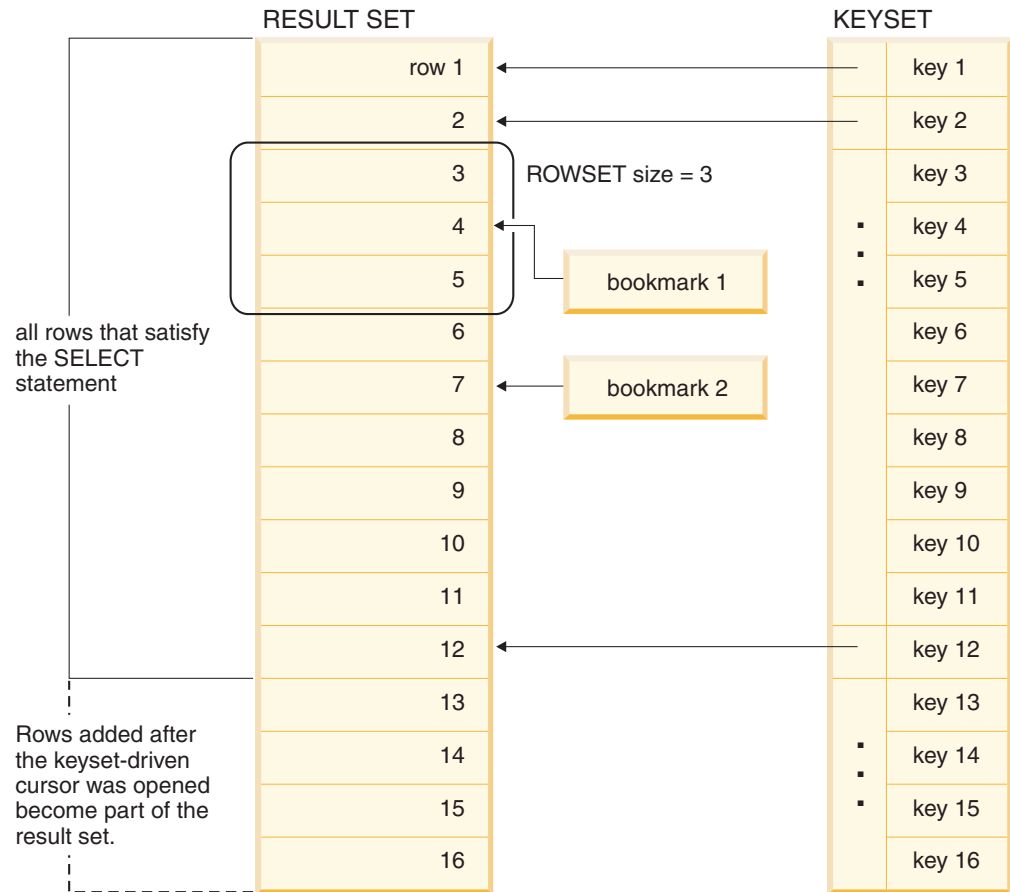
The subset of rows from the result set that is returned after each fetch. The application indicates the size of the rowset before the first fetch of data, and can modify the size before each subsequent fetch. Each call to `SQLFetch()`, `SQLFetchScroll()`, or `SQLExtendedFetch()` populates the rowset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



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.

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.

Rowset retrieval examples in CLI applications

The rowset is a cache that holds predefined rows of data that are returned in the result set.

Partial rowset example

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.

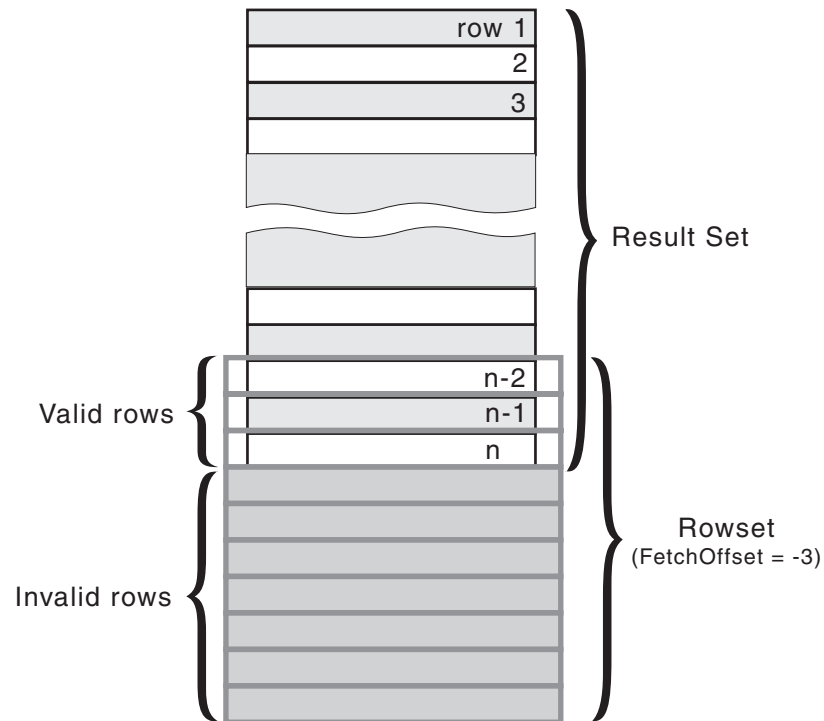


Figure 5. Partial rowset example

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.

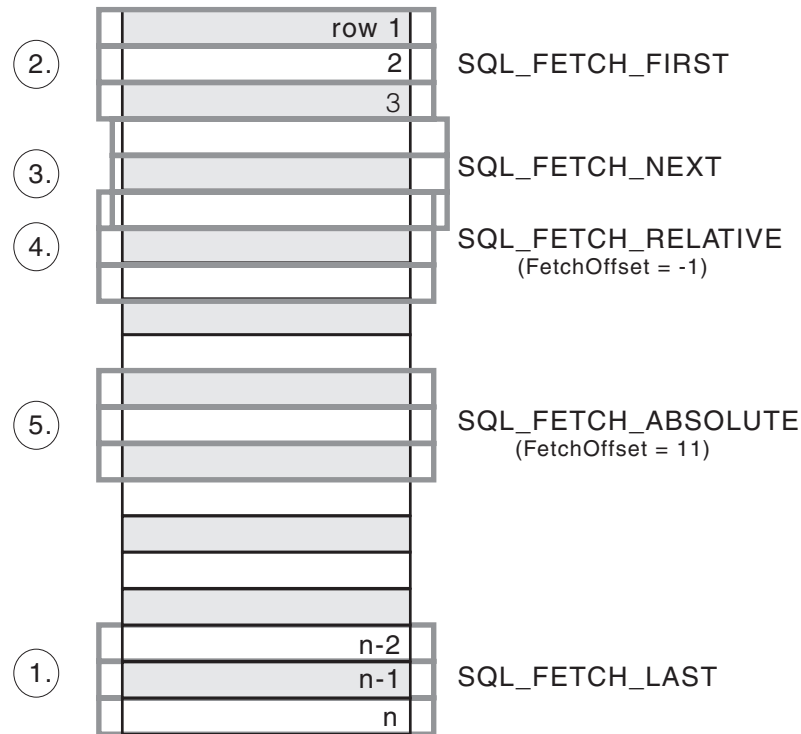


Figure 6. Example of retrieving rowsets

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.

Before you begin

Before you retrieve results, ensure that 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 resource usage 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 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.

Column binding in CLI applications

The column binding is associating columns in a result set to C data type variables, and associating LOB columns to LOB locators or LOB file references.

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.

If a CLI application does not provide an output buffer for a LOB column using the function `SQLBindCol()` the IBM data server client will, by default, request a LOB locator on behalf of the application for each LOB column in the result sets.

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 or XML column. 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(), 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, 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.

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.

Before you begin

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

About this task

CLI allows an application to specify a rowset for a non-scrollable or scrollable cursor that spans more than one row at a time.

Procedure

To effectively work with a rowset, an application should perform the following steps:

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 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. Set up a variable that will store the number of rows returned. Declare a variable of type `SQLINTEGER` 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()`:

```
/* ... */

SQLINTEGER 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 <code>SQLFetch()</code>)	Page Down
Previous rowset	SQL_FETCH_PRIOR	Page Up
Rowset starting on next row	SQL_FETCH_RELATIVE with <i>FetchOffset</i> set to 1	Line Down
Rowset starting on previous row	SQL_FETCH_RELATIVE with <i>FetchOffset</i> set to -1	Line Up
Rowset starting on a specific row	SQL_FETCH_ABSOLUTE with <i>FetchOffset</i> 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 <i>FetchOffset</i> 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.

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.

Before you begin

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).

- 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);
```

- Issue an SQL SELECT statement.
- Execute the SQL SELECT statement.
- Bind the result set using either column-wise or row-wise binding.
- Fetch a rowset of rows from the result set.
 - 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 */
```

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

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.

Before you begin

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:

- 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 required 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, `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`.

Retrieving bulk data with bookmarks using `SQLBulkOperations()` in CLI applications

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

Before you begin

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

About this task

Bookmarks in 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.

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()`.

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 resource usage 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 resource usage, 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 7 is a logical view of column-wise binding.

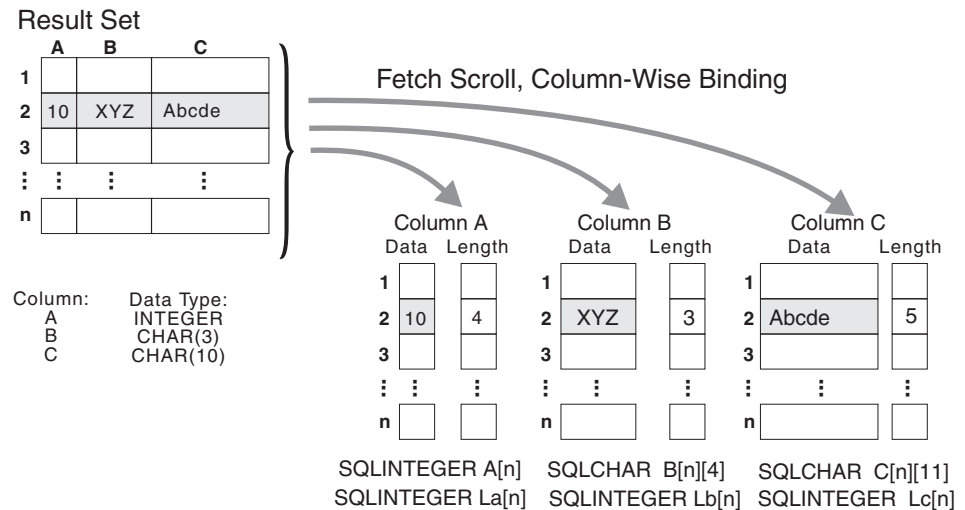


Figure 7. Column-wise binding

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 8 gives a pictorial view of row-wise binding.

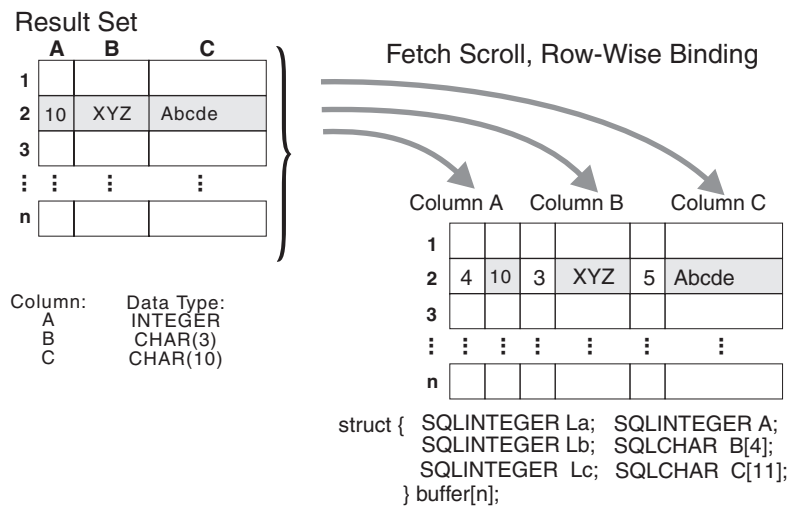


Figure 8. Row-wise binding

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.

Before you begin

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, 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, 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.

Retrieving array data in CLI applications using row-wise binding

When retrieving data, you might 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.

Before you begin

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 can allocate the example 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.

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, 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.

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()`, 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()`.

Before you begin

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

About this task

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 in step 1.
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 value 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 as required.

Data retrieval in pieces in CLI applications

Typically, an application might 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). 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.

Typically, an application might 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.

A call to `SQLGetData()` (which is called after `SQLFetch()`) 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.

Fetching LOB data with LOB locators in CLI applications

There are many cases where an application needs to fetch a large object value by referencing a large object locator (LOB locator).

An example is used to demonstrate 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.

Before you begin

Before fetching LOB data with LOB locators, ensure that 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, 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 ) ;

```

XML data retrieval in CLI applications

For XML data, when you use SQLBindCol() to bind columns in a query result set to application variables, you can specify the data type of the application variables as SQL_C_BINARY, SQL_C_CHAR, SQL_C_DBCHAR or SQL_C_WCHAR.

When you select data from XML columns in a table, the output data is in the serialized string format.

For XML data, when you use SQLBindCol() to bind columns in a query result set to application variables, you can specify the data type of the application variables as SQL_C_BINARY, SQL_C_CHAR, SQL_C_DBCHAR or SQL_C_WCHAR. When retrieving a result set from an XML column, it is recommended that you bind your application variable to the SQL_C_BINARY type. Binding to character types can result in possible data loss resulting from code page conversion. Data loss can occur when characters in the source code page cannot be represented in the target code page. Binding your variable to the SQL_C_BINARY C type avoids these issues.

XML data is returned to the application as internally encoded data. CLI determines the encoding of the data as follows:

- If the C type is SQL_C_BINARY, CLI returns the data in the UTF-8 encoding scheme.
- If the C type is SQL_C_CHAR or SQL_C_DBCHAR, CLI returns the data in the application code page encoding scheme.

- If the C type is SQL_C_WCHAR, CLI returns the data in the UCS-2 encoding scheme.

The database server performs an implicit serialization of the data before returning it to the application. You can explicitly serialize the XML data to a specific data type by calling the XMLSERIALIZE function. Implicit serialization is recommended, however, because explicitly serializing to character types with XMLSERIALIZE can introduce encoding issues.

The following example shows how to retrieve XML data from an XML column into a binary application variable.

```
char xmlBuffer[10240];
// xmlBuffer is used to hold the retrieved XML document
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

length = sizeof (xmlBuffer);
SQLExecute (hStmt, "SELECT deptdoc FROM dept WHERE id='001'", SQL_NTS);
SQLBindCol (hStmt, 1, SQL_C_BINARY, xmlBuffer, &length, NULL);
SQLFetch (hStmt);
SQLCloseCursor (hStmt);
// xmlBuffer now contains a valid XML document encoded in UTF-8
```

Inserting data

Inserting bulk data with bookmarks using SQLBulkOperations() in CLI applications

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

Before you begin

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

About this task

Bookmarks in 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.

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.

Before you begin

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

Note: The CLI **LOAD** interface to the IBM DB2 **LOAD** utility is not supported when accessing Informix database servers.

About this task

Note: Starting from Version 9.7, Fix Pack 4, this feature can also be used with the CLI async processing feature.

- Unlike the IBM DB2 **LOAD** utility, the CLI **LOAD** utility does not load data directly from an input file. Instead, if required, 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.
- If the prepared SQL statement for inserting data contains a `SELECT` clause, parameter markers are not supported.
- The prepared SQL statement for inserting data must include parameter markers for all columns in the target table, unless a `fullselect` is used instead of the `VALUES` clause in the `INSERT` statement.
- The insertion of data is non-atomic because the load utility precludes atomicity. **LOAD** might 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 CLI function can be called until `SQL_USE_LOAD_OFF` is set (see Step 3):

- `SQLBindParameter()`
 - `SQLExecute()`
 - `SQLExtendedBind()`
 - `SQLParamOptions()`
 - `SQLSetStmtAttr()`
2. Create a structure of type `db2LoadStruct` and specify the required load options through this structure. Set the `SQL_ATTR_LOAD_INFO` statement attribute to a pointer to this structure.
 3. Optional: The `ANYORDER` file type modifier option of the `LOAD` API can potentially increase the performance of the load. Set the statement attribute `SQL_ATTR_LOAD_MODIFIED_BY` in `SQLSetStmtAttr()` to specify the file type modifier option `ANYORDER`.

For example, the following call specifies the `anyorder` file type modifier for the CLI `LOAD`:

```
char *filemod="anyorder";
SQLSetStmtAttr (hstmt, SQL_ATTR_LOAD_MODIFIED_BY,
                (SQLPOINTER) filemod, SQL_NTS);
```

4. Issue `SQLExecute()` on the prepared SQL statement for the data to be inserted. The `INSERT` SQL statement can be a `fullselect` which allows data to be loaded from a table using the `SELECT` statement. With a single execution of the `INSERT` statement, all of the data from the `SELECT` is loaded. The following example shows how a `fullselect` statement loads data from one table into another:

```
SQLPrepare (hStmt,
            (SQLCHAR *) "INSERT INTO tableB SELECT * FROM tableA",
            SQL_NTS);
SQLExecute (hStmt);
```

5. 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).

6. Optional: After the CLI **LOAD** operation, you can query the number of rows that were affected by it by using the following statement attributes:
 - `SQL_ATTR_LOAD_ROWS_COMMITTED_PTR`: A pointer to an integer that represents the total number of rows processed. This value equals the number of rows successfully loaded and committed to the database, plus the number of skipped and rejected rows.
 - `SQL_ATTR_LOAD_ROWS_DELETED_PTR`: A pointer to an integer that represents the number of duplicate rows deleted.
 - `SQL_ATTR_LOAD_ROWS_LOADED_PTR`: A pointer to an integer that represents the number of rows loaded into the target table.
 - `SQL_ATTR_LOAD_ROWS_READ_PTR`: A pointer to an integer that represents the number of rows read.
 - `SQL_ATTR_LOAD_ROWS_REJECTED_PTR`: A pointer to an integer that represents the number of rows that could not be loaded.
 - `SQL_ATTR_LOAD_ROWS_SKIPPED_PTR`: A pointer to an integer that represents the number of rows skipped before the CLI **LOAD** operation began.

To use the statement attributes to query the number of rows affected by the CLI **LOAD**, the application must call `SQLSetStmtAttr` before the CLI **LOAD**, and pass a pointer to the memory location where the value will be stored.

For example, after you turn on CLI **LOAD** by calling `SQLSetStmtAttr` and specify the statement attribute `SQL_ATTR_USE_LOAD_API` as in step 1, before executing the `INSERT` to do the CLI **LOAD**, you can call `SQLSetStmtAttr` to pass a pointer to the memory location where the value will be stored.

```
int *rowsLoaded;
int *rowsDeleted;

rowsLoaded = (int *)malloc(sizeof(int));
if (rowsLoaded == NULL)
{
    // Handle any memory allocation failure by malloc
}
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_LOAD_ROWS_LOADED_PTR, rowsLoaded,
SQL_IS_POINTER);

rowsDeleted = (int *)malloc(sizeof(int));
if (rowsDeleted == NULL)
{
    // Handle any memory allocation failure by malloc
}
rc = SQLSetStmtAttr(hstmt, SQL_ATTR_LOAD_ROWS_DELETED_PTR, rowsDeleted,
SQL_IS_POINTER);
```

After the CLI **LOAD**, you can retrieve the statement attribute values as follows:

```
printf("\n Value of SQL_ATTR_LOAD_ROWS_LOADED_PTR is %d", *rowsLoaded);
printf("\n Value of SQL_ATTR_LOAD_ROWS_DELETED_PTR is %d", *rowsDeleted);
```

You can also retrieve the statement attribute values by calling `SQLGetStmtAttr`, as shown in the following example. Note that you must call `SQLSetStmtAttr` to pass a pointer to the memory location where the value will be stored before you issue the `INSERT` statement for the CLI **LOAD**.

```
int *pStmtAttrValue;

rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_LOAD_ROWS_LOADED_PTR,
                    &pStmtAttrValue,
                    sizeof(pStmtAttrValue),
                    NULL);
printf("\n Value of SQL_ATTR_LOAD_ROWS_LOADED_PTR is %d", *pStmtAttrValue);
```

```

rc = SQLGetStmtAttr(hstmt,
                    SQL_ATTR_LOAD_ROWS_DELETED_PTR,
                    &pStmtAttrValue,
                    sizeof(pStmtAttrValue),
                    NULL);
printf("\n Value of SQL_ATTR_LOAD_ROWS_DELETED_PTR is %d", *pStmtAttrValue);

```

XML column inserts and updates in CLI applications

When you update or insert data into XML columns of a table, the input data must be in the serialized string format. For XML data, when you use `SQLBindParameter()` to bind parameter markers to input data buffers, you can specify the data type of the input data buffer as `SQL_C_BINARY`, `SQL_C_CHAR`, `SQL_C_DBCHAR` or `SQL_C_WCHAR`.

When you update or insert data into XML columns of a table, the input data must be in the serialized string format.

For XML data, when you use `SQLBindParameter()` to bind parameter markers to input data buffers, you can specify the data type of the input data buffer as `SQL_C_BINARY`, `SQL_C_CHAR`, `SQL_C_DBCHAR` or `SQL_C_WCHAR`.

When you bind a data buffer that contains XML data as `SQL_C_BINARY`, CLI processes the XML data as internally encoded data. This is the preferred method because it avoids additional resource usage and potential data loss of character conversion when character types are used.

Important: If the XML data is encoded in an encoding scheme and CCSID other than the application code page encoding scheme, you must include internal encoding in the data and bind the data as `SQL_C_BINARY` to avoid character conversion.

When you bind a data buffer that contains XML data as `SQL_C_CHAR`, `SQL_C_DBCHAR` or `SQL_C_WCHAR`, CLI processes the XML data as externally encoded data. CLI determines the encoding of the data as follows:

- If the C type is `SQL_C_WCHAR`, CLI assumes that the data is encoded as UCS-2.
- If the C type is `SQL_C_CHAR` or `SQL_C_DBCHAR`, CLI assumes that the data is encoded in the application code page encoding scheme.

If you want the database server to implicitly parse the data before storing it in an XML column, the parameter marker data type in `SQLBindParameter()` should be specified as `SQL_XML`.

Implicit parsing is recommended, because explicit parsing of a character type with `XMLPARSE` can introduce encoding issues.

The following example shows how to update XML data in an XML column using the recommended `SQL_C_BINARY` type.

```

char xmlBuffer[10240];
integer length;

// Assume a table named dept has been created with the following statement:
// CREATE TABLE dept (id CHAR(8), deptdoc XML)

// xmlBuffer contains an internally encoded XML document that is to replace
// the existing XML document
length = strlen (xmlBuffer);

```



```

SQLPrepare (hStmt, "UPDATE dept SET deptdoc = ? WHERE id = '001'", SQL_NTS);
SQLBindParameter (hStmt, 1, SQL_PARAM_INPUT, SQL_C_BINARY, SQL_XML, 0, 0,
                  xmlBuffer, 10240, &length);
SQLExecute (hStmt);

```

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.

Before you begin

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.

Updating bulk data with bookmarks using SQLBulkOperations() in CLI applications

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

Before you begin

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

About this task

Bookmarks in 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.

Deleting bulk data with bookmarks using SQLBulkOperations() in CLI applications

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

Before you begin

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

About this task

Bookmarks in 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.

Calling stored procedures from 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.

Before you begin

Before calling a stored procedure, ensure that you have initialized your CLI application.

About this task

If the stored procedure being called is uncataloged, ensure that it does not call any of the CLI schema functions. Calling CLI schema functions from uncataloged stored procedures is not supported.

The CLI schema functions are: `SQLColumns()`, `SQLColumnPrivileges()`, `SQLForeignKeys()`, `SQLPrimaryKeys()`, `SQLProcedureColumns()`, `SQLProcedures()`, `SQLSpecialColumns()`, `SQLStatistics()`, `SQLTables()`, and `SQLTablePrivileges()`.

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. CLI 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 (?)";
```

or

```
SQLCHAR *stmt = (SQLCHAR *)"CALL OUT_LANGUAGE (:language)";
```

Remember: To use named parameter markers (for example, :language), you must explicitly enable named parameter processing by setting the **EnableNamedParameterSupport** configuration keyword to TRUE.

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`.

Note: The CALL procedure statement does not support an array input for a parameter marker using the `SQL_ATTR_PARAMSET_SIZE` attribute.

6. Execute the CALL procedure statement using `SQLExecuteDirect()`, 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 `SQLExecuteDirect()` 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 columns 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.

Results

Once you have executed the CALL statement, you can retrieve result sets from the stored procedure if applicable.

Note:

The numeric month and day parts of a DATETIME data type value will appear to be reversed in procedure result sets that are returned to DB2 CLI applications if the values are not returned in ISO format. For example, this can happen if a local format is used instead. To ensure that DATETIME data type value information is correctly interpreted by a client application, the procedures should be bound to the database with a locale-independent DATETIME format such as ISO. For example:

```
db2set DB2_SQLROUTINE_PREPOPTS="DATETIME ISO"
```

Note:

CLI packages are automatically bound to databases when the databases are created or upgraded.

Resultsets from Anonymous blocks

CLI will be prepared for resultsets whenever a SQL statement that starts with BEGIN, but not BEGIN COMPOUND, is sent by the application. CLI will interpret the returned cursor from the server and allow application to retrieve the resultsets.

Example 1: Using SQLExecDirect

```
opt caller on
opt echo on

quickc 1 1 sample

SQLAllocStmt 1 1
getmem 1 1 SQL_C_LONG

SQLExecDirect 1 "drop table t1" -3
SQLExecDirect 1 "create table t1 (c1 int)" -3
SQLExecDirect 1 "insert into t1 values (10)" -3
SQLExecDirect 1 "insert into t1 values (20)" -3
SQLExecDirect 1 "insert into t1 values (30)" -3

SQLExecDirect 1 "begin declare c1 cursor with return to client with hold
for select c1 from t1; end" -3
SQLBindCol 1 1 sql_c_long 1
FetchAll 1

SQLFreeStmt 1 SQL_DROP
SQLTransact 1 1 SQL_COMMIT

killenv 1
```

CLI stored procedure commit behavior

The commit behavior of SQL statements, both in a CLI client application and in the called stored procedure running on a DB2 server, depends on the commit combinations applied in the application and the stored procedure.

The possible combinations and the resulting commit behavior are described in the following table.

Table 8. CLI Stored procedure commit behavior

CLI client	Stored procedure	Commit behavior
autocommit on	autocommit on	All successfully executed SQL statements in the stored procedure are committed, even if other SQL statements in the stored procedure fail and an error or warning SQLCODE is returned to the CALL statement.
autocommit on	autocommit off	If the stored procedure returns an SQLCODE ≥ 0 , all successfully executed SQL statements in the stored procedure are committed. Otherwise, all SQL statements in the stored procedure are rolled back.
autocommit on	manual commit	All successfully executed SQL statements in the stored procedure that are manually committed will not be rolled back, even if an error SQLCODE is returned to the CALL statement. Note: If the stored procedure returns an SQLCODE ≥ 0 , any successfully executed SQL statements in the stored procedure that occur after the last manual commit will be committed; otherwise, they will be rolled back to the manual commit point.
autocommit off	autocommit on	All successfully executed SQL statements in the stored procedure are committed and will not be rolled back, even if an error SQLCODE is returned to the CALL statement. In addition, all uncommitted and successfully executed SQL statements in the CLI client application up to and including the CALL statement are committed. Note: Exercise caution when using this commit combination in a multi-SQL statement client-side transaction, because the transaction cannot be fully rolled back after the CALL statement has been issued.
autocommit off	autocommit off	If the stored procedure returns an SQLCODE ≥ 0 , all successfully executed SQL statements in the stored procedure will be committed when the transaction that includes the CALL statement is committed. Otherwise, all SQL statements in the stored procedure will be rolled back when the transaction that includes the CALL statement is rolled back.
autocommit off	manual commit	All successfully executed SQL statements in the stored procedure that are manually committed will not be rolled back, even if an error SQLCODE is returned to the CALL statement. In addition, all uncommitted and successfully executed SQL statements in the CLI client application up to the CALL statement are committed. Note: If the stored procedure returns an SQLCODE ≥ 0 , any successfully executed SQL statements within the stored procedure that occur after the last manual commit will be committed; otherwise, they will be rolled back to the manual commit point. Note: Exercise caution when using this commit combination in a multi-SQL statement client-side transaction, because the transaction cannot be fully rolled back after the CALL statement has been issued.

Creating static SQL by using CLI/ODBC static profiling

You can use the CLI/ODBC static profiling feature to replace dynamic SQL statements with static SQL statements, potentially improving runtime performance and security through the package-based authorization mechanism.

About this task

When you run an application with prebound static SQL statements, dynamic registers that control the dynamic statement behavior have no effect on the statements that are converted to static SQL statements.

If an application issues Data Definition Language (DDL) statements for objects that are referenced in subsequent Data Manipulation Language (DML) statements, all of these statements are in the capture file. The CLI/ODBC static package binding tool command, **db2cap**, attempts to bind them. The bind attempt is successful only for a database management system (DBMS) that supports the VALIDATE RUN bind option. If the DBMS does not support the VALIDATE RUN bind option, the application should not use static profiling.

You can edit the capture file to add, change, or remove SQL statements, based on application-specific requirements.

The following restrictions apply when you are running an application during the profiling session:

- An SQL statement must have successfully run (generated a positive SQLCODE value) for it to be captured in a profiling session. In a statement-matching session, unmatched dynamic statements continue to execute as dynamic CLI/ODBC 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 to be different from COL=1. Use parameter markers in place of literals to improve match hits.

Not all dynamic CLI/ODBC calls can be captured and grouped into a static package. Possible reasons why calls are not captured and grouped are as follows:

- The application does not regularly free environment handles. During a capture session, statements that are captured under a particular environment handle are written to the capture file or files only when that environment handle is freed.
- 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 registry variable values. These statements are not recorded. There is a limited capability in match mode to detect dynamic SET SQLID and SET SCHEMA statements and suspend running static statements accordingly. However, other SET statements and subsequent SQL statements that depend on the registry variables being set might not behave correctly.
- The application issues DML statements. Depending on application complexities and the nature of these statements, they might not be matched, or they might not execute correctly at run time.

Because dynamic and static SQL are different, always verify the behaviour of the application in static match mode. Furthermore, static SQL does not offer improved runtime performance over dynamic SQL for all statements. If testing shows that static execution decreases performance for a particular statement, you can force that statement to be dynamically executed by removing the statement from the capture file. In addition, static SQL, unlike dynamic SQL, might require occasional rebinding of packages to maintain performance, particularly if the database objects that are referred to in the packages frequently change. If CLI/ODBC static profiling does not fit the type of application that you are running, there are other

programming methods that you can use to obtain the benefits of static SQL, such as embedded SQL and stored procedures.

Procedure

To create static SQL statements from dynamic SQL statements:

1. Profile the application by capturing all the dynamic SQL statements that the application issues. 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 data source in the `db2cli.ini` configuration file before running the application:

- `StaticMode = CAPTURE`
- `StaticPackage = qualified_package_name`
- `StaticCapFile = capture_file_name`

An example of the settings follows:

```
[DSN1]
StaticMode = CAPTURE
StaticPackage = MySchema.MyPkg
StaticCapFile = E:\Shared\MyApp.cpt
```

Note: For the **StaticPackage** keyword, ensure that you specify a schema name (MySchema in the previous sample). If you do not specify a schema, the database object name that you provide is considered to be the container name instead of the package name, and the package name is blank.

The resulting static profile takes the form of a text-based *capture file*, containing information about the SQL statements that are captured.

The previous example file yields the following results for Data Source Name 1 (DSN1):

- Capture mode will be used.
- The package will be named MySchema.MyPkg
- The capture file, MyApp.cpt, will be saved in the E:\Shared\ directory.

Until you change the **StaticMode** keyword to a value other than CAPTURE, such as DISABLED(which turns off static capture mode), each subsequent run of the application captures SQL statements and appends them to the capture file (MyApp.cpt in the example. Only unique SQL statements are captured, however; duplicate executions are ignored.

2. Optional: To generate a CLI/ODBC static profiling log file, set the **StaticLogFile**CLI/ODBC configuration keyword. This file contains useful information to determine the state of the statement capturing process.
3. Run the application. Unique SQL statements are captured in the capture file. Duplicate statements are ignored.
4. Disable static capture mode by setting the **StaticMode**CLI/ODBC configuration keyword to DISABLED or by removing the keywords that you set in the first step.
5. From the Command Line Processor, issue the **db2cap** command. The **db2cap** command generates a static package that is based on the capture file. If the **db2cap** command does not return a message indicating successful completion, a statement in the capture file could not be statically bound. Remove the failing statement from the capture file, and run the **db2cap** command again.
6. Make a copy of the capture file that you processed with the **db2cap** command available to each user of the application. You can give the file to the users or, if all users use the same client platform, place a read-only copy of this file in a network directory that is accessible to all users.

7. Enable your application for dynamic-to-static SQL statement mapping, known as static match mode, by setting the following configuration keywords:

- `StaticMode = MATCH`
- `StaticCapFile = capture_file_name`

An example of settings follows:

```
[DSN1]
StaticMode = MATCH
StaticCapFile = E:\Shared\MyApp.cpt
```

8. Optional: Set the CLI/ODBC **StaticLogFile** configuration keyword to log useful information such as how many statements were matched (therefore statically executed) and how many statements were unmatched (therefore dynamically executed) during a match session. You should use this information to verify that static profiling in match mode is yielding an acceptable match ratio.

9. Run the application.

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.

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.
STMTEXT	Statement string
STMSTYPE	There are 3 possible values: <ul style="list-style-type: none"> • <code>SELECT_CURSOR_WITHHOLD</code>: SELECT statement using a withhold cursor • <code>SELECT_CURSOR_NOHOLD</code>: SELECT statement using a nohold cursor • <code>OTHER</code>: non-SELECT statements
CURSOR	Cursor name declared for the SELECT statement

Field	Description
INVARnn	<p>Description of the n-th input variable</p> <p>The 7 comma-separated fields refer to:</p> <ol style="list-style-type: none"> 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	<p>Description of the n-th output variable for the SELECT statement. The comma-separated fields follow the same convention as in INVARs.</p>

Considerations for mixing embedded SQL and CLI

You can use 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 CLI catalog functions and maximize the portion of the application's processing where performance is critical.

You can use 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 CLI catalog functions and maximize the portion of the application's processing where performance is critical. In order to mix the use of CLI and embedded SQL, the application must comply with the listed rules:

- All connection management and transaction management must be performed completely using either 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 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 CLI APIs, notably, a null connection.
- Query statement processing cannot straddle CLI and embedded SQL interfaces for the same statement. For example, the application cannot open a cursor using embedded SQL, and then call the CLI `SQLFetch()` function to retrieve row data.

Since CLI permits multiple connections, the `SQLSetConnection()` function must be called before executing any embedded SQL. This allows the application to explicitly specify the connection under which the embedded SQL processing is performed.

If the CLI application is multithreaded and also makes embedded SQL calls or DB2 API calls, then each thread must have a DB2 context.

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.

About this task

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()`.

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

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 previously.

Procedure

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, use `SQLFreeHandle()` in your CLI applications so that they conform to the latest standards.

Handle freeing in CLI applications

Before 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.

Environment handle

Before 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 before 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, 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, CLI frees all the automatically generated descriptors associated with that handle.

Chapter 7. 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.

Before you begin

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`.

Chapter 8. Trusted connections through DB2 Connect

Some DB2 database servers support trusted contexts. A *trusted context* allows the database administrator to, among other things, define conditions under which a client application will be allowed to create a trusted connection. A *trusted connection* is allowed to do things that a normal connection cannot.

There are two types of trusted connection, implicit and explicit. When you create a connection, whether you get an explicit trusted connection, an implicit trusted connection, or a regular connection depends on whether you ask for a trusted connection and whether the connection meets the criteria defined in the trusted context on the server, as summarized in Table 9.

Table 9. What type of connections result from different combinations of actions

	The connection meets the server's criteria for being trusted	The connection does not meet the server's criteria for being trusted
You request that the connection be trusted	Explicit trusted connection	Regular connection and warning SQL20360W (SQLSTATE 01679) is returned.
You do not request that the connection be trusted	Implicit trusted connection	Regular connection

An *implicit trusted connection* is identical to a regular connection except that it grants temporary role privileges to the user while they are using the connection. The role privileges that are granted (if any) are specified in the trusted context that caused the connection to be trusted.

Implicit trusted connections can be created by any application that connects using DB2 Connect. Implicit trusted connections are made and used in the same way that regular connections are made and used. This means that no code changes are necessary for an existing application to take advantage of implicit trusted connections as long as the application connects through DB2 Connect.

An *explicit trusted connection* grants temporary role privileges to the user the same way that an implicit trusted connection does. In addition, an explicit trusted connection lets you change the authorization ID used when performing actions across that connection. Changing the authorization ID on an explicit trusted connection is referred to as *switching users*. The authorization IDs to which you can switch and whether a given authorization ID requires a password when switching to it are defined as part of the trusted context that allowed the trusted connection to be created.

User switching can significantly reduce the processing usage of sharing a connection among several users, especially for user names that do not require a password because in that case the database server does not authenticate the authorization ID. When using the feature, however, you must be very certain that your application does not allow switching to an authorization ID without validating and authenticating that authorization ID. Otherwise you are creating a security hole in your system.

Explicit trusted connections can be created and the user can be switched when connecting through DB2 Connect using CLI or JDBC, including XA established connections. Creating an explicit trusted connection and switching users requires setting special connection attributes. This means that existing applications will need to be modified in order to take advantage of explicit trusted connections.

Other than the differences just mentioned, you can use a trusted connection (whether implicit or explicit) the same way you would use a regular connection. You must be certain, however, to explicitly disconnect an explicit trusted connection when you are done with it, even if it is in a broken or disconnected state. Otherwise resources used by the connection might not be released. This is not a problem with implicit trusted connections.

Note:

1. Explicit trusted connections should not use CLIENT authentication. This does not apply to implicit trusted connections.
2. Applications using explicit trusted connections should be run on secure machines which are password protected and accessible only to authorized personnel. This does not apply to implicit trusted connections.

Creating and terminating a trusted connection through CLI

If the database server you are connecting to is configured to allow it, you can create an explicit trusted connection when connecting through CLI.

Before you begin

This procedure assumes that you are not using an XA transaction manager. If you are using an XA transaction manager you only need to make sure that the transaction manager is configured to set the configuration value TCTX to TRUE when it calls `xa_open`. If that is done then any connection that can be an explicit trusted connection will be. To verify that a connection is an explicit trusted connection see step 3.

- The database that you are connecting to must support trusted contexts.
- A trusted context must be defined that will recognize the client as being trustable.
- You must know the system authorization ID that is specified in the trusted context. The system authorization ID of a trusted connection is the authorization ID you provide to the server as a user name when creating the connection. For your connection to be trusted by a particular trusted context the system authorization ID must be the one specified in that trusted context. Ask your security administrator for a valid system authorization ID and the password for that ID.

About this task

The examples in these instructions use the C language and assume that `conn` is a pointer to a valid, but unconnected, connection handle. The variable `rc` is assumed to have a data type of SQLRETURN.

Procedure

1. In addition to setting any connection attributes that you would set for a regular connection, set the connection attribute `SQL_ATTR_USE_TRUSTED_CONTEXT` to `SQL_TRUE` with a call to the `SQLSetConnectAttr` function.


```
rc = SQLSetConnectAttr(
    conn,
    SQL_ATTR_USE_TRUSTED_CONTEXT, SQL_TRUE, SQL_IS_INTEGER
);
```

2. Connect to the database as you would for a regular connection, for example by calling the `SQLConnect` function. Use the system authorization ID as the user name and its password as the password. Be sure to check for errors and warnings, especially those listed in table Table 10.

Table 10. Errors indicating failure to create a trusted connection

SQLCODE	SQLSTATE	Meaning
SQL20360W	01679	The connection could not be established as a trusted connection. It was established as a regular connection instead.

If no errors or warnings tell you differently, then the connection is established and is an explicit trusted connection.

3. Optional: You can verify that an established connection is an explicit trusted connection by checking the value of the connection attribute `SQL_ATTR_USE_TRUSTED_CONTEXT` using the `SQLGetConnectAttr` function. If it is set to `SQL_TRUE` the connection is an explicit trusted connection.
4. When you are finished using the connection you must be very careful to explicitly disconnect it, even if it is in a broken or disconnected state. If you do not explicitly disconnect an explicit trusted connection some of the resources used by the connection might not be released.

Results

Note:

1. Explicit trusted connections should not use `CLIENT` authentication. This does not apply to implicit trusted connections.
2. Applications using explicit trusted connections should only be run on secure computers which are password protected and accessible only to authorized personnel. This does not apply to implicit trusted connections.

Switching users on a trusted connection through CLI

You can switch users on an explicit trusted connection through the command line interface (CLI).

For a description of what it means to switch users using a trusted connection see the topic in the related links.

Before you begin

- The connection must have been successfully created as an explicit trusted connection.
- The explicit trusted connection must not be in a transaction.
- The trusted context that allowed the explicit trusted connection to be created must be configured to allow switching to the authorization ID you are switching to.

About this task

The examples in these instructions use the C language and assume that `conn` is a pointer to a connected explicit trusted connection. The variable `rc` is assumed to have a data type of `SQLRETURN`. The variable `newuser` is assumed to be a pointer

to a character string holding the authorization ID of the user you want to switch to. The variable *passwd* is assumed to be a pointer to a character string containing the password for that authorization ID.

Procedure

1. Call the `SQLSetConnectAttr` function to set the `SQL_ATTR_TRUSTED_CONTEXT_USERID` attribute. Set it to the authorization ID you want to switch to.

```
rc = SQLSetConnectAttr(
    conn,
    SQL_ATTR_TRUSTED_CONTEXT_USERID, newuser, SQL_NTS
);
//Check for errors
```

Be sure to check for errors and warnings, especially those listed in table Table 11.

Table 11. Errors indicating failure to set a new authorization ID when switching users

SQLCODE	Meaning
CLI0106E	The connection is not connected.
CLI0197E	The connection is not a trusted connection.
CLI0124E	There is a problem with the value provided. Check that it is not null, or not too long, for example.
CLI0196E	The connection is involved in a unit of work that prevents it from switching users. To be able to switch users the connection must not be in a transaction.

2. Optional: (This step is optional unless the trusted context that allowed this trusted connection requires a password for the authorization ID you are switching to.) Call the `SQLSetConnectAttr` function to set the `SQL_ATTR_TRUSTED_CONTEXT_PASSWORD` attribute. Set it to the password for the new authorization ID.

```
rc = SQLSetConnectAttr(
    conn,
    SQL_ATTR_TRUSTED_CONTEXT_PASSWORD, passwd, SQL_NTS
);
//Check for errors
```

Be sure to check for errors and warnings, both those listed in table Table 11 and those listed in table Table 12.

Table 12. Errors indicating failure to set a password when switching users

SQLCODE	Meaning
CLI0198E	The attribute <code>SQL_ATTR_TRUSTED_CONTEXT_USERID</code> has not yet been set.

3. Proceed as with a regular connection. If you are using an XA transaction manager the user switch is attempted as part of the next request, otherwise the user switch is attempted just before initiating the next function call that accesses the database (`SQLExecDirect` for example). In either case, in addition to the errors and warnings you would normally check for, be sure to check for the errors listed in Table 13 on page 137. The errors in Table 13 on page 137 indicate that the user switch failed.

Table 13. Errors indicating failure to switch users

SQLCODE	Meaning
SQL1046N	The trusted context that allowed this trusted connection is not configured to allow switching to the authorization ID you are trying to switch to. You will not be able to switch to that authorization ID until the trusted context is changed.
SQL30082N	The password provided is not correct for the authorization ID you are switching to.
SQL0969N with a native error of -20361	There is some database level constraint that prevent you from switching to the user.

If the user switch fails the connection will be in an unconnected state until you successfully switch to another user. You can switch users on a trusted connection in an unconnected state but cannot access the database server with it. A connection in an unconnected state will remain in that state until you successfully switch users on it.

What to do next

Note:

1. **Important:** Switching users without supplying a password bypasses the database server's authentication. Your application must not allow a switch to an authorization ID without a password unless that application has already validated and authenticated that authorization ID. To do otherwise creates a security hole.
2. Specifying a NULL value for the `SQL_ATTR_TRUSTED_CONTEXT_USERID` attribute is equivalent to specifying the trusted context system authorization ID (the user id used when the explicit trusted connection was created).
3. When you successfully set the value of the `SQL_ATTR_TRUSTED_CONTEXT_USERID` connection attribute on an explicit trusted connection the connection is immediately reset. The result of resetting is as if a new connection were created using the original connection attributes of that connection. This reset happens even if the value you set the connection attribute to is the system authorization ID or NULL or the same value that the attribute currently holds.
4. If the `SQL_ATTR_TRUSTED_CONTEXT_PASSWORD` attribute is set, the password will be authenticated during the switch user processing, even if the trusted context that allowed the trusted connection doesn't require authentication on a switch user for that authorization ID. This results in unnecessary processing time. This rule doesn't apply to the trusted context system authorization ID. If the trusted context system authorization ID doesn't require authentication when you switch to it then it is not authenticated even if a password is provided.

Chapter 9. Descriptors in CLI applications

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.

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 CLI has performed any required conversion.
- If the parameter is used as output, this describes the SQL data before 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 CLI performs any required data conversion to the application's C data types.

The only difference between four types of descriptors 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.

The list of the header fields are:

- 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). 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.

The fields in a descriptor record describe a column or parameter, and occur once in each descriptor record. The fields in a descriptor record are:

- 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
- SQL_DESC_CARDINALITY
- SQL_DESC_CARDINALITY_PTR

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 14. 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)	<ul style="list-style-type: none">• 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)
SQL_DESC_CARDINALITY_PTR	(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. When 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.

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.

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, 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.

Descriptor allocation and freeing

Descriptors are allocated in one of two ways: Implicitly allocated descriptors, and Explicitly allocated descriptors. When using implicitly allocated descriptors 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. When using explicitly allocated descriptors an application can explicitly allocate application descriptors. It is not possible, however, to allocate implementation descriptors.

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)

The following example 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`. The following example 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 increased resource usage. 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`.

Descriptor manipulation with descriptor handles in CLI applications

Descriptors can be manipulated using descriptor handles or with CLI functions that do not use descriptor handles. This topic describes accessing descriptors through descriptor handles.

Descriptors can be manipulated using descriptor handles or with 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 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 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 CLI.)

For example, all of the descriptor fields 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).

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.

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.

Chapter 10. 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 CLI catalog functions instead.

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 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 CLI catalog functions are:

- `SQLColumnPrivileges()`
- `SQLColumns()`
- `SQLExtendedProcedures()`
- `SQLExtendedProcedureColumns()`
- `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 might 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.

Note: By default, Informix database servers return schema information (such as table names, and column names) in the system catalog in lowercase. This is different from DB2 data servers which return schema information in upper case.

Some of the catalog functions result in execution of fairly complex queries. It is recommended that the application save the information returned rather than making repeated calls to get the same information.

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.

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.

Input arguments to catalog functions may be treated as ordinary arguments or pattern value arguments. 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.

Some catalog functions accept pattern values on some of their input arguments. For example, `SQLColumnPrivileges()` treats *SchemaName* and *TableName* as ordinary arguments and *ColumnName* as a pattern value. Refer to the "Function Arguments" section of the specific catalog function to see if a particular input argument accepts pattern values.

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 );
    /* ... */
}

```

Chapter 11. Programming hints and tips for CLI applications

These hints and tips can help you tune and improve the logic of your CLI applications.

- KEEP_DYNAMIC support
- Common connection attributes
- Common statement attributes
- Reusing statement handles
- Binding and SQLGetData()
- Limiting use of catalog functions
- Column names of function generated result sets
- CLI-specific functions loaded from ODBC applications
- Global dynamic statement caching
- Data insertion and retrieval optimization
- Large object data optimization
- Case sensitivity of object identifiers
- SQLDriverConnect() versus SQLConnect()
- Turning off statement scanning
- Holding cursors across rollbacks
- Preparing compound SQL sub-statements
- User-defined types casting
- Deferred prepare to reduce network flow

KEEP_DYNAMIC behavior refers to the server's ability to keep a dynamic statement in a prepared state, even after a commit has been performed. This behavior eliminates the need for the client to prepare the statement again, the next time the statement is executed. Some CLI/ODBC applications on the client might improve their performance by taking advantage of the **KEEP_DYNAMIC** behavior on servers that are DB2 for z/OS and OS/390 Version 7 and later. Complete the listed steps to enable **KEEP_DYNAMIC** behavior:

1. Enable the dynamic statement cache on the DB2 for z/OS and OS/390 server (see the DB2 for z/OS and OS/390 server documentation).
2. Bind the db2cli.pk.bnd file on your DB2 for Linux, UNIX, and Windows client with the **KEEP_DYNAMIC** and **COLLECTION** options. The example shows how to bind db2cli.pk.bnd, creating a collection named **KEEP_DYNAMIC**:
 - db2 connect to *database_name* user *userid* using *password*
 - db2 bind db2cli.pk.bnd SQLERROR CONTINUE BLOCKING ALL KEEP_DYNAMIC YES COLLECTION KEEP_DYNAMIC GRANT PUBLIC
 - db2 connect reset
3. Inform the client that the **KEEP_DYNAMIC** bind option is enabled for your collection by performing either of the listed examples:

- Set the CLI/ODBC configuration keywords in the db2cli.ini file:
KeepDynamic = 1, CurrentPackageSet = collection name created in Step 2.

For example:

```
[dbname]  
KeepDynamic=1  
CurrentPackageSet=KEEP_DYNAMIC
```

- Set the SQL_ATTR_KEEPPDYNAMIC and SQL_ATTR_CURRENT_PACKAGE_SET connection attributes in the CLI/ODBC application. For example:

```
SQLSetConnectAttr(hDbc,
                  SQL_ATTR_KEEP_DYNAMIC,
                  (SQLPOINTER) 1,
                  SQL_IS_UINTEGER );

SQLSetConnectAttr(hDbc,
                  SQL_ATTR_CURRENT_PACKAGE_SET,
                  (SQLPOINTER) "KEEPPDYN",
                  SQL_NTS);
```

See the DB2 for z/OS and OS/390 server documentation for further information about **KEEPPDYNAMIC** behavior and configuration.

Common connection attributes

The listed connection attributes can be set by CLI applications:

- SQL_ATTR_AUTOCOMMIT - Generally this attribute should be set to SQL_AUTOCOMMIT_OFF, because 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 must choose an isolation level that maximizes concurrency, yet ensures data consistency.

Common statement attributes

The listed statement attributes might be set by 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 FIRST n ROWS ONLY" clauses to the statement. For versions of DB2 for OS/390 before Version 7 and any DBMS that does not support the "FETCH FIRST n ROWS ONLY" clause, the full result set is still generated at the server using the "OPTIMIZE FOR n ROWS" clause, however 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 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 appropriate 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 - CLI allows the isolation level to be set at the statement level, however, it is recommended 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 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 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 might need to be released by calling the `SQLFreeStmt()` function. Also, statement attributes previously set on a statement handle (for example, `SQL_ATTR_PARAMSET_SIZE`) must be explicitly reset, otherwise they might 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 for more information). Use `SQLGetData()` when the data value is large variable-length data that:

- must be received in pieces, or
- might not need to be retrieved.

Limiting use of catalog functions

Catalog functions, such as `SQLTables()`, force the 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 might be developed and tested against a data source with hundreds of tables, it might 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 might change as the ODBC and CLI standards evolve. The *position* of the columns, however, will not change.

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

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 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 CLI statement handle (HSTMT). The CLI functions can then be called directly from the shared library or DLL, passing the HSTMT argument where required.

Global dynamic statement caching

DB2 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 `SQLGetPosition()` 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 resource utilization of the application transferring data through 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 uppercase 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 listed two tables are created,

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

two tables will exist, MYTABLE and YourTable

Both of the statements are equivalent:

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

The second statement in the example will fail with TABLE NOT FOUND because there is no table named YOURTABLE:

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

All 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 CLI to prompt the user for the connection information.

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

Turning off statement scanning

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 must be set to SQL_NOSCAN_ON at the connection level so that CLI does not perform a scan for vendor escape clauses.

Holding cursors across rollbacks

Applications that must deal with complex transaction management issues might benefit from establishing multiple concurrent connections to the same database. Each connection in 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; because the cursors are in separate transactions, a rollback on one statement does not affect the cursors of the other statements.

However, using multiple connections might 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 want to use (for example to insert) the portion of the LOB values represented by the LOB locators, you must 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 can just use the LOB locators directly. However, the LOB locators are lost 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 might 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 might be prepared before the BEGIN COMPOUND statement, and then executed within the compound statement.

This also simplifies error handling because 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 listed 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 listed SQL statement was then issued:

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

This statement fails 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 because 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 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 an application generates queries with very small answer sets, because the resource utilization for requests and replies flowing over the network represents a large percentage of the processing time. 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 occur.

Reduction of network flows with CLI array input chaining

CLI array input chaining is a feature that, when enabled, causes requests for the execution of prepared statements to be held and queued at the client until the chain is ended. Once the chain has been ended, all of the chained `SQLExecute()` requests at the client are then sent to the server in a single network flow.

CLI array input chaining is a feature that, when enabled, causes requests for the execution of prepared statements to be held and queued at the client until the chain is ended. Once the chain has been ended, all of the chained `SQLExecute()` requests at the client are then sent to the server in a single network flow.

The following sequence of events (presented as pseudocode) is an example of how CLI array input chaining can reduce the number of network flows to the server:

```
SQLPrepare (statement1)
SQLExecute (statement1)
SQLExecute (statement1)
/* the two execution requests for statement1 are sent to the server in
two network flows */

SQLPrepare (statement2)

/* enable chaining */
SQLSetStmtAttr (statement2, SQL_ATTR_CHAINING_BEGIN)

SQLExecute (statement2)
SQLExecute (statement2)
SQLExecute (statement2)

/* end chaining */
```

```
SQLSetStmtAttr (statement2, SQL_ATTR_CHAINING_END)
```

```
/* the three execution requests for statement2 are sent to the server  
in a single network flow, instead of three separate flows */
```

If `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO` is returned when setting `SQL_ATTR_CHAINING_END`, then at least one statement in the chain of statements returned `SQL_ERROR` or `SQL_SUCCESS_WITH_INFO` when it was executed. Use the CLI diagnostic functions `SQLGetDiagRec()` and `SQLGetDiagField()` to retrieve information about what has caused the error or warning.

Restriction: DB2 CLI does not support array input chaining for compound SQL (compiled) or compound SQL (inlined) statements.

Chapter 12. Unicode CLI applications

The CLI Unicode application must connect to the database with either the `SQLConnectW()` or `SQLDriverConnectW()` function.

The connecting to the database with either the `SQLConnectW()` or `SQLDriverConnectW()` function ensures that the CLI driver considers Unicode as the preferred method of communication between itself and the database.

There are two main areas of support for 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.

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 Unicode data. The CLI/ODBC driver considers all Unicode data exchanged with the application to be UCS-2 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 are described using the new types. Conversion is allowed between `SQL_C_WCHAR` and `SQL_CHAR`, `SQL_VARCHAR`, `SQL_LONGVARCHAR` and `SQL_CLOB`, as well as with the graphic data types.

Note: UCS-2 is a fixed-length character encoding scheme that uses 2 bytes to represent each character. When referring to the number of characters in a UCS-2 encoded string, the count is simply the number of `SQLWCHAR` elements needed to store the string.

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 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, 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.

Unicode functions (CLI)

CLI Unicode functions accept Unicode string arguments in place of ANSI string arguments.

CLI Unicode functions accept Unicode string arguments in place of ANSI string arguments. The Unicode string arguments must be in UCS-2 encoding (native-endian format). 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 CLI). The following list of CLI functions are available in both ANSI and Unicode versions:

- `SQLBrowseConnect`
- `SQLColAttribute`
- `SQLColAttributes`
- `SQLColumnPrivileges`
- `SQLColumns`
- `SQLConnect`
- `SQLCreateDb`
- `SQLDataSources`
- `SQLDescribeCol`
- `SQLDriverConnect`
- `SQLDropDb`
- `SQLError`
- `SQLExecDirect`
- `SQLExtendedPrepare`
- `SQLExtendedProcedures`
- `SQLExtendedProcedureColumns`
- `SQLForeignKeys`
- `SQLGetConnectAttr`
- `SQLGetConnectOption`
- `SQLGetCursorName`
- `SQLGetDescField`
- `SQLGetDescRec`
- `SQLGetDiagField`
- `SQLGetDiagRec`
- `SQLGetInfo`
- `SQLGetPosition`
- `SQLGetStmtAttr`
- `SQLNativeSQL`
- `SQLPrepare`
- `SQLPrimaryKeys`
- `SQLProcedureColumns`
- `SQLProcedures`
- `SQLReloadConfig`

- SQLSetConnectAttr
- SQLSetConnectOption
- SQLSetCursorName
- SQLSetDescField
- SQLSetStmtAttr
- SQLSpecialColumns
- SQLStatistics
- SQLTablePrivileges
- SQLTables

Unicode functions that have arguments which are always the length of strings interpret these arguments as the number of SQLWCHAR elements needed to store the string. For functions that return length information for server data, the display size and precision are again described in terms of the number of SQLWCHAR elements used to store them. When the length (transfer size of the data) can refer to string or non-string data, it is interpreted as the number of bytes needed to store the data.

For example, `SQLGetInfoW()` will still take the length as the number of bytes, but `SQLExecDirectW()` will use the number of SQLWCHAR elements. Consider a single character from the UTF-16 extended character set (UTF-16 is an extended character set of UCS-2; Microsoft Windows 2000 and Microsoft Windows XP use UTF-16). Microsoft Windows 2000 will use two `SQL_C_WCHAR` elements, which is equivalent to 4 bytes, to store this single character. The character therefore has a display size of 1, a string length of 2 (when using `SQL_C_WCHAR`), and a byte count of 4. 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 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.

Unicode function calls to ODBC driver managers

ODBC-compliant applications can access a DB2 database through the CLI/ODBC driver in one of two ways: linking to the CLI/ODBC driver library or linking to the ODBC driver manager library.

ODBC-compliant applications can access a DB2 database through the CLI/ODBC driver in one of two ways: linking to the CLI/ODBC driver library or linking to the ODBC driver manager library. This topic discusses CLI applications that link to the ODBC driver manager library.

- Direct access - An application links to the CLI/ODBC driver library and makes calls to exported CLI/ODBC functions. Unicode applications accessing the CLI/ODBC driver directly should access and perform transactions against the database using the CLI Unicode functions, and use `SQLWCHAR` buffers with the

understanding that all Unicode data is UCS-2. To identify itself as a Unicode application, the application must connect to the database using either `SQLConnectW()` or `SQLDriverConnectW()`.

- Indirect access - An application links to an ODBC driver manager library and makes calls to standard ODBC functions. The ODBC driver manager then loads the CLI/ODBC driver and calls exported ODBC functions on behalf of the application. The data passed to the CLI/ODBC driver from the application might be converted by the ODBC driver manager. An application identifies itself to an ODBC driver manager as a Unicode application by calling `SQLConnectW()` or `SQLDriverConnectW()`.

When connecting to a data source, the ODBC driver manager checks to see if the requested driver exports the `SQLConnectW()` function. If the function is supported, the ODBC driver is considered a Unicode driver, and all subsequent calls in the application to ODBC functions are routed to the functions' Unicode equivalents (identified by the 'W' suffix; for example, `SQLConnectW()`) by the ODBC driver manager. If the application calls Unicode functions, no string conversion is necessary, and the ODBC driver manager calls the Unicode functions directly. If the application calls ANSI functions, the ODBC driver manager converts all ANSI strings to Unicode strings before calling the equivalent Unicode function.

If an application calls Unicode functions, but the driver does not export `SQLConnectW()`, then the ODBC driver manager routes any Unicode function calls to their ANSI equivalents. All Unicode strings are converted by the ODBC driver manager to ANSI strings in the application's code page before calling the equivalent ANSI function. This might result in data loss if the application uses Unicode characters which cannot be converted to the application's code page.

Various ODBC driver managers use different encoding schemes for Unicode strings, depending on the operating system:

Table 15. Unicode string encoding schemes by operating system

Driver manager	Operating system	
	Microsoft Windows	Linux and UNIX
Microsoft ODBC Driver Manager	UTF-16*	not applicable
unixODBC Driver Manager	UCS-2	UCS-2
DataDirect Connect for ODBC Driver Manager	UTF-16*	UTF-8
* UTF-16 is a superset of UCS-2 and therefore is compatible		

Chapter 13. 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.

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. CLI applications can be written to use various transaction managers:

- DB2 as transaction manager
- Process-based XA-compliant transaction program monitor
- Host and IBM Power Systems™ database servers

Note: There is no specific DB2 CLI/ODBC client configuration required when connecting to a host or IBM Power Systems database server, although the machine running DB2 Connect might require certain configuration settings to enable running multisite update mode against the host.

ConnectType CLI/ODBC configuration keyword

Controls whether the application is to operate in a remote or distributed unit of work.

db2cli.ini keyword syntax:

ConnectType = 1 | 2

Default setting:

Remote unit of work.

Equivalent environment or connection attribute:

SQL_ATTR_CONNECTTYPE

Usage notes:

This option allows you to specify the default connect type. The options are:

- 1 = Remote unit of work. Multiple concurrent connections, each with its own commit scope. The concurrent transactions are not coordinated. This is the default.
- 2= Distributed unit of work. Coordinated connections where multiple databases participate under the same distributed unit of work.

The first connection determines the connect type for all other connections that are allocated under the same environment handle.

This keyword takes precedence over the environment or connection attribute.

DB2 as transaction manager in CLI applications

CLI/ODBC applications can use DB2 itself as the Transaction Manager (DB2 TM) to coordinate distributed transactions against all IBM database servers.

Configuration of DB2 as transaction manager

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 configurations must be applied:

- The `SQL_ATTR_CONNECTTYPE` environment attribute must be set. This attribute 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.
 - `SQL_COORDINATED_TRANS` - supports multiple databases per transaction semantics. A coordinated transaction is one in which commits or rollbacks among multiple database connections are coordinated. Setting `SQL_ATTR_CONNECTTYPE` to this value corresponds to Type 2 `CONNECT` in embedded SQL.

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. However, 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.

All connections on an environment handle must have the same `SQL_ATTR_CONNECTTYPE` setting. An environment cannot have a mixture of concurrent and coordinated connections. The type of the first connection will determine the type of all subsequent connections. `SQLSetEnvAttr()` will return an error if an application attempts to change the connect type while there is an active connection.

- If `SQL_ATTR_CONNECTTYPE` is set to `SQL_COORDINATED_TRANS`, 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 function `SQLEndTran()` must be used in a multisite update environment when DB2 is acting as the transaction manager.

Application flows in concurrent and coordinated transactions

Figure 9 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 10 on page 168 shows the same statements being executed on two `SQL_COORDINATED_TRANS` connections ('A' and 'B'), and the scope of a coordinated distributed transaction.

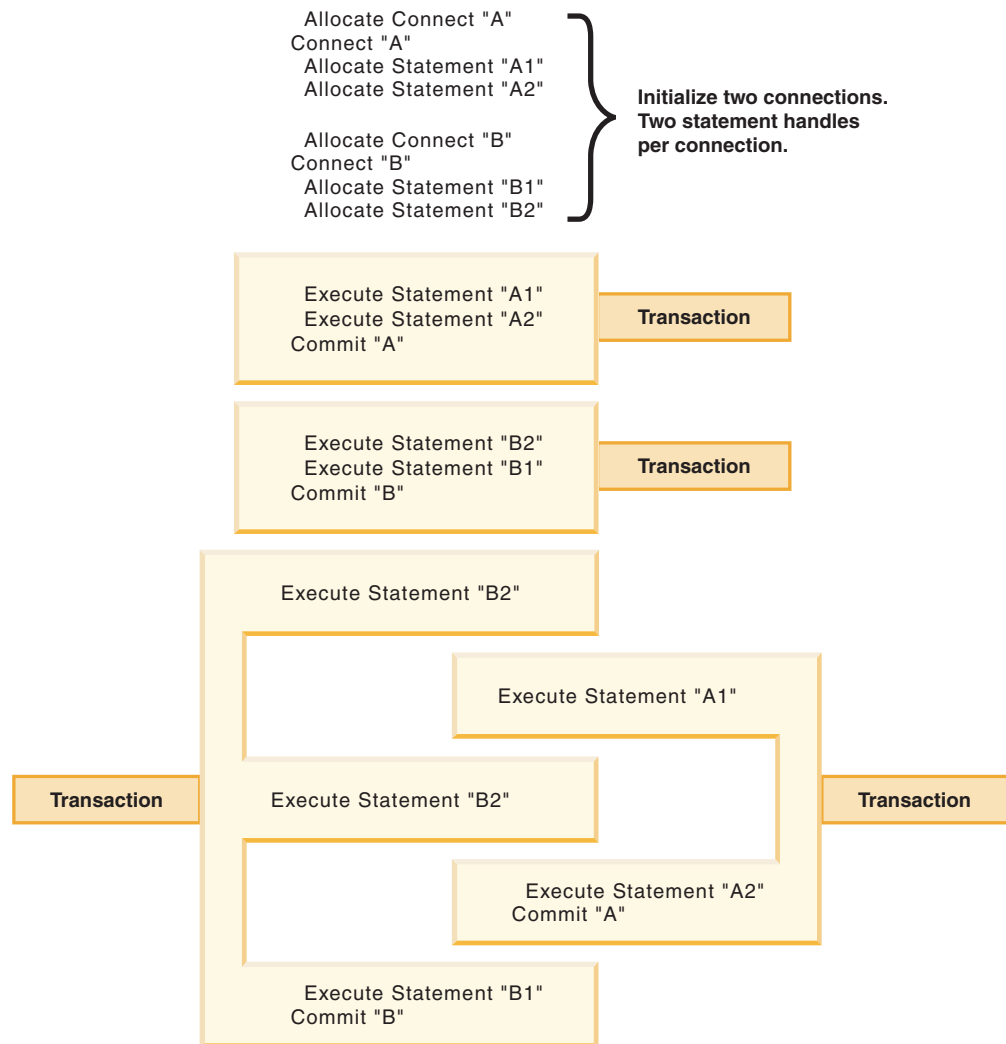


Figure 9. Multiple connections with concurrent transactions

```

Allocate Environment
Set Environment Attribute
(SQL_ATTR_CONNECTTYPE)

Allocate Connect "A"
Connect "A"
(SQL_COORDINATED_TRANS)

Allocate Statement "A1"
Allocate Statement "A2"

Allocate Connect "B"
Connect "B"
(SQL_COORDINATED_TRANS)

Allocate Statement "B1"
Allocate Statement "B2"

```

**Initialize two connections.
Two statement handles
per connection.**

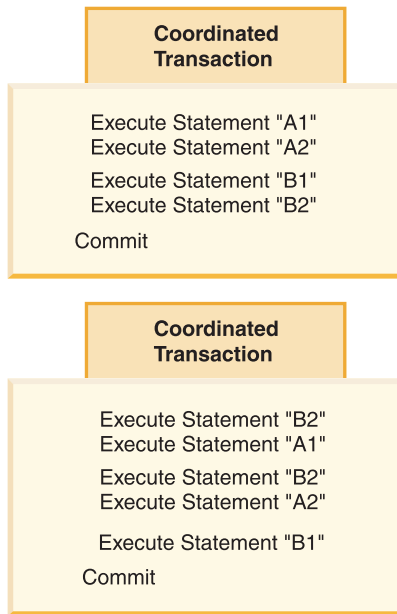


Figure 10. Multiple connections with coordinated transactions

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.

Process-based XA-compliant Transaction Program Monitor (XA TP) programming considerations for CLI applications

Process-based XA TPs, for example CICS®, start 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 CLI/ODBC applications under this environment.

Process-based XA TPs, for example CICS, start 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 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

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 tasks 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.

Chapter 14. Asynchronous execution of CLI functions

CLI can run a subset of CLI functions asynchronously. For these functions, the CLI driver returns control to the application after calling the function but before that function has finished executing.

CLI can run a subset of CLI functions asynchronously. For these functions, the CLI driver returns control to the application after calling the function but before that function has finished executing.

Asynchronous execution is possible for those functions that normally send a request to the server and then wait for a response. The functions return `SQL_STILL_EXECUTING` each time they are called until they are finished running, at which point they return a different value (for example, `SQL_SUCCESS`). Rather than waiting for a response, a function executing asynchronously returns control to the application. The application can then perform other tasks and poll the function until a return code other than `SQL_STILL_EXECUTING` is returned. Refer to the `SQL_ATTR_ASYNC_ENABLE` connection or statement attribute for a list of functions that can be executed asynchronously.

In order for an application to run CLI functions asynchronously, the application must include following function calls:

1. A call to the function `SQLGetInfo()` with the `SQL_ASYNC_MODE` option to ensure support for asynchronous calls.
2. A call to `SQLSetConnectAttr()` or `SQLSetStmtAttr()` with the `SQL_ATTR_ASYNC_ENABLE` attribute to enable asynchronous calls once it has been established that there is support for asynchronous calls.
3. A call to a function that supports asynchronous execution and polling of the asynchronous function. When the application calls a function that can be run asynchronously, one of two things can happen:
 - If the function will not benefit from being run asynchronously, CLI can decide to run it synchronously and return the normal return code (other than `SQL_STILL_EXECUTING`). In this case the application runs as it would if the asynchronous mode had not been enabled.
 - CLI will perform some minimal processing (such as checking the arguments for errors), then pass the statement on to the server. Once this quick processing is complete a return code of `SQL_STILL_EXECUTING` is returned to the application.

Functions that can be called during asynchronous execution

Once a function has been called asynchronously, only the original function, `SQLAllocHandle()`, `SQLCancel()`, `SQLGetDiagField()`, or `SQLGetDiagRec()` can be called on the statement or the connection associated with *StatementHandle*, until the original function returns a code other than `SQL_STILL_EXECUTING`. Any other function called on *StatementHandle* or the connection associated with *StatementHandle* returns `SQL_ERROR` with an `SQLSTATE` of `HY010` (Function sequence error.).

Diagnostic information while a function is running asynchronously

SQLGetDiagField() returns the following values when it is called on a statement handle that has an asynchronous function executing:

- The values of SQL_DIAG_CURSOR_ROW_COUNT, SQL_DIAG_DYNAMIC_FUNCTION, SQL_DIAG_DYNAMIC_FUNCTION_CODE, and SQL_DIAG_ROW_COUNT header fields are undefined.
- SQL_DIAG_NUMBER header field returns 0.
- SQL_DIAG_RETURN_CODE header field returns SQL_STILL_EXECUTING.
- All record fields return SQL_NO_DATA.

SQLGetDiagRec() always returns SQL_NO_DATA when it is called on a statement handle that has an asynchronous function executing.

Cancelling the asynchronous function call

The application can issue a request to cancel any function that is running asynchronously by calling SQLCancel(). A function that has already finished executing cannot be cancelled.

The return code from the SQLCancel() call indicates whether the cancel request was received, not whether the execution of the asynchronous function was stopped.

The only way to tell if the function was cancelled is to call it again, using the original arguments.

- If the cancel was successful, the function will return SQL_ERROR and an SQLSTATE of HY008 (Operation was Canceled.).
- If the cancel was not successful, the function will return a value other than SQL_ERROR with an SQLSTATE of HY008. For example, the function might return SQL_STILL_EXECUTING.

Executing functions asynchronously in CLI applications

Executing functions asynchronously in CLI applications is part of the larger task of programming with CLI.

The task of enabling asynchronous functions and working with those functions involves ensuring that asynchronous execution is supported, initializing the application for asynchronous execution, and working with the functions to take advantage of asynchronous execution.

Before you begin

Before you begin setting up your CLI application for asynchronous execution, you must allocate an environment handle and a connection handle. This is part of the task of initializing your CLI application.

About this task

Note: Starting from Version 9.7, Fix Pack 4, this feature can also be used with the CLI load processing feature.

An application can have at most 1 active function running in asynchronous mode on any one connection. If asynchronous mode is enabled at the connection level, all statements already allocated, as well as future statement handles allocated on the connection will be enabled for asynchronous execution.

Procedure

1. Call `SQLGetInfo()` with *InfoType* `SQL_ASYNC_MODE` to ensure that functions can be called asynchronously. For example:

```
/* See what type of Asynchronous support is available. */
rc = SQLGetInfo( hdbc, /* Connection handle */
                SQL_ASYNC_MODE, /* Query the support available */
                &buffer, /* Store the result in this variable */
                4,
                &outlen);
```

The call to the `SQLGetInfo()` function will return one of the following values:

- `SQL_AM_STATEMENT`: asynchronous execution can be turned on or off at a statement level.
 - `SQL_AM_CONNECTION`: asynchronous execution can be turned on or off at a connection level.
 - `SQL_AM_NONE`: asynchronous execution is not supported. Your application cannot be set up for asynchronous execution. This will be returned for one of two reasons:
 - The datasource itself does not support asynchronous execution.
 - The CLI/ODBC configuration keyword `ASYNCENABLE` has been specifically set to disable asynchronous execution.
2. Set the `SQL_ATTR_ASYNC_ENABLE` attribute using `SQLSetStmtAttr()` or `SQLSetConnectAttr()` to enable your application for asynchronous execution if the return value from `SQLGetInfo()` is either `SQL_AM_STATEMENT` or `SQL_AM_CONNECTION`.

- If the return value is `SQL_AM_STATEMENT`, set `SQL_ATTR_ASYNC_ENABLE` to `SQL_ASYNC_ENABLE_ON` using `SQLSetStmtAttr()`. For example:

```
/* Set statement level asynchronous execution on */
rc = SQLSetStmtAttr( hstmt, /* Statement handle */
                    SQL_ATTR_ASYNC_ENABLE,
                    (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
                    0);
```

- If the return value is `SQL_AM_CONNECTION`, set the `SQL_ATTR_ASYNC_ENABLE` to `SQL_ASYNC_ENABLE_ON` using `SQLSetConnectAttr()`. For example:

```
/* Set connection level asynchronous execution on */
rc = SQLSetConnectAttr( hstmt, /* Connection handle */
                       SQL_ATTR_ASYNC_ENABLE,
                       (SQLPOINTER) SQL_ASYNC_ENABLE_ON,
                       0);
```

3. Call a function that supports asynchronous execution and poll the asynchronous function. Refer to the `SQL_ATTR_ASYNC_ENABLE` connection or statement attribute for a list of functions that can be executed asynchronously.

The application determines whether the function has completed by calling it repeatedly with the same arguments it used to call the function the first time. A return code of `SQL_STILL_EXECUTING` indicates it is not yet finished, any other value indicates it has completed. The value other than `SQL_STILL_EXECUTING` is the same return code it would have returned if it had executed synchronously.

The following example demonstrates a common while loop that takes both possible outcomes into account:

```
while ( (rc = SQLExecDirect(hstmt, sqlstmt, SQL_NTS) ) == SQL_STILL_EXECUTING)
{
    /* Other processing can be performed here, between each call to
    * see if SQLExecDirect() has finished running asynchronously.
    * This section will never run if CLI runs the function
    * synchronously.
    */
}
/* The application continues at this point when SQLExecDirect() */
/* has finished running. */
```

Chapter 15. Multithreaded CLI applications

CLI supports concurrent execution of threads on the following platforms: AIX, HP-UX, Linux, Solaris and Windows.

CLI supports concurrent execution of threads on the following platforms:

- AIX
- HP-UX
- Linux
- Solaris
- Windows

On any other platform that supports threads, CLI is guaranteed to be thread safe by serializing all threaded access to the database. In other words, applications or stored procedures that use CLI can be invoked multiple times and at the same time.

Note: If you are writing applications that use 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 CLI serializes all function calls, only one thread may be executing a 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 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 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 CLI.

Applications that use multiple connections, and are executing statements that may take some time to execute, should consider executing 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 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 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`.

Note:

1. Thread safety for handles only applies for 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.

Application model for multithreaded CLI applications

A multithreaded CLI application allows parallel execution of tasks under a single process.

The typical application model for multithreaded CLI application consists of following attributes:

- 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 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 CLI controls access to its resources, the application resources such as bound columns and parameter buffers are not controlled by 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 keyword is set, the application will wait forever.

The application model discussed earlier avoids this problem by not sharing application resources between threads once a thread starts executing on a connection.

Mixed multithreaded CLI applications

It is possible for a multi-threaded application to mix CLI calls with DB2 API calls and embedded SQL.

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:

CLI Calls first

The 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 CLI to allocate and manage all contexts. Use CLI to allocate all connection handles and to perform all connections. Call the `SQLSetConnect()` function in each thread before calling any embedded SQL. DB2 APIs can be called after any CLI function has been called in the same thread.

DB2 API or embedded SQL calls first

The 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 `sqlAttachToCtx()` which will explicitly attach the thread to a context, or by calling any 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 before calling 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 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.

Chapter 16. 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 CLI and ODBC support vendor escape clauses as defined by X/Open.

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 CLI and ODBC support vendor escape clauses as defined by X/Open.

Currently, escape clauses are used extensively by ODBC to define SQL extensions. 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.

CLI used both the standard and shorthand syntax for escape clauses. The standard syntax has been deprecated (although 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 per current ODBC standard.

Shorthand escape clause syntax

The format of an escape clause definition is:

```
{ extended SQL text }
```

to define the listed 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 `SELECT * FROM EMPLOYEE WHERE HIREDATE={d '1994-03-29'}` statement can be used to issue a query against the **EMPLOYEE** table.

CLI will translate the select 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, CLI will translate the 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 `SELECT Name FROM Customers WHERE Growth LIKE '1_\'%'\{escape '\}'` statement would select all of the values from `Name` that have values in `Growth` only between 10% and 19%.

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 can 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 can 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 through SQLBindParameter(). CLI will return the return code as the procedure's return value if ?= is present in the escape clause. If ?= is not present, and if the stored procedure return code is not SQL_SUCCESS, then the application can retrieve diagnostics, including the SQLCODE, using the SQLGetDiagRec() and SQLGetDiagField() functions. CLI supports literals as procedure arguments, however vendor escape clauses must be used. For example, the CALL storedproc ('aaaa', 1) statement would not succeed, but {CALL storedproc ('aaaa', 1)} statement would. If a parameter is an output parameter, it must be a parameter marker.

For example, CLI will translate the 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, CLI will translate the 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`.

Extended scalar functions for CLI applications

The following functions are defined by ODBC using vendor escape clauses. Each function can be called using the escape clause syntax, or calling the equivalent DB2 function.

The following functions are defined by ODBC using vendor escape clauses. Each function can 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
- Date and time functions
- System functions
- Conversion function

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 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 CLI and defined by ODBC using vendor escape clauses.

- Character string literals used as arguments to scalar functions must be bounded by single quotation marks.

- 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 16. String scalar functions

String scalar function	Description	Servers that support the function
ASCII(<i>string_exp</i>)	Returns the ASCII code value of the leftmost character of <i>string_exp</i> as an integer.	DB2 for Linux, UNIX, and Windows
CHAR(<i>code</i>)	Returns the character that has the ASCII code value specified by <i>code</i> . The value of <i>code</i> should be between 0 and 255; otherwise, the return value is null.	DB2 for Linux, UNIX, and Windows
CONCAT(<i>string_exp1</i> , <i>string_exp2</i>)	Returns a character string that is the result of concatenating <i>string_exp2</i> to <i>string_exp1</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
DIFFERENCE(<i>string_exp1</i> , <i>string_exp2</i>)	Returns an integer value indicating the difference between the values returned by the SOUNDEX function for <i>string_exp1</i> and <i>string_exp2</i> .	DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i
INSERT(<i>string_exp1</i> , <i>start</i> , <i>length</i> , <i>string_exp2</i>)	Returns a character string where <i>length</i> number of characters beginning at <i>start</i> has been replaced by <i>string_exp2</i> which contains <i>length</i> characters.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
LCASE(<i>string_exp</i>)	Converts all uppercase characters in <i>string_exp</i> to lowercase.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE
LEFT(<i>string_exp</i> , <i>count</i>)	Returns the leftmost <i>count</i> of characters of <i>string_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
LENGTH(<i>string_exp</i>)	Returns the number of characters in <i>string_exp</i> , excluding trailing blanks and the string termination character. Note: Trailing blanks are included for DB2 Server for VM and VSE.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

Table 16. String scalar functions (continued)

String scalar function	Description	Servers that support the function
LOCATE(<i>string_exp1</i> , <i>string_exp2</i> [, <i>start</i>])	Returns the starting position of the first occurrence of <i>string_exp1</i> within <i>string_exp2</i> . The search for the first occurrence of <i>string_exp1</i> begins with first character position in <i>string_exp2</i> unless the optional argument, <i>start</i> , is specified. If <i>start</i> is specified, the search begins with the character position indicated by the value of <i>start</i> . The first character position in <i>string_exp2</i> is indicated by the value 1. If <i>string_exp1</i> is not found within <i>string_exp2</i> , the value 0 is returned.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
LTRIM(<i>string_exp</i>)	Returns the characters of <i>string_exp</i> with the leading blanks removed.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
REPEAT(<i>string_exp</i> , <i>count</i>)	Returns a character string composed of <i>string_exp</i> repeated <i>count</i> times.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
REPLACE(<i>string_exp1</i> , <i>string_exp2</i> , <i>string_exp3</i>)	Replaces all occurrences of <i>string_exp2</i> in <i>string_exp1</i> with <i>string_exp3</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS,
RIGHT(<i>string_exp</i> , <i>count</i>)	Returns the rightmost count of characters of <i>string_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
RTRIM(<i>string_exp</i>)	Returns the characters of <i>string_exp</i> with trailing blanks removed.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
SOUNDEX(<i>string_exp1</i>)	Returns a four character code representing the sound of <i>string_exp1</i> . Note that different data sources use different algorithms to represent the sound of <i>string_exp1</i> .	DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i
SPACE(<i>count</i>)	Returns a character string consisting of <i>count</i> spaces.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
SUBSTRING(<i>string_exp</i> , <i>start</i> , <i>length</i>)	Returns a character string that is derived from <i>string_exp</i> beginning at the character position specified by <i>start</i> for <i>length</i> characters.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

Table 16. String scalar functions (continued)

String scalar function	Description	Servers that support the function
<i>UCASE(string_exp)</i>	Converts all lowercase characters in <i>string_exp</i> to uppercase.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

Numeric functions

The numeric functions in this section are supported by 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 17. Numeric scalar functions

Numeric scalar function	Description	Servers that support the function
<i>ABS(numeric_exp)</i>	Returns the absolute value of <i>numeric_exp</i>	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>ACOS(double_exp)</i>	Returns the arccosine of <i>double_exp</i> as an angle, expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>ASIN(double_exp)</i>	Returns the arcsine of <i>double_exp</i> as an angle, expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>ATAN(double_exp)</i>	Returns the arctangent of <i>double_exp</i> as an angle, expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

Table 17. Numeric scalar functions (continued)

Numeric scalar function	Description	Servers that support the function
<i>ATAN2(double_exp1, double_exp2)</i>	Returns the arctangent of <i>x</i> and <i>y</i> coordinates specified by <i>double_exp1</i> and <i>double_exp2</i> , as an angle expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>CEILING(numeric_exp)</i>	Returns the smallest integer greater than or equal to <i>numeric_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>COS(double_exp)</i>	Returns the cosine of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>COT(double_exp)</i>	Returns the cotangent of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>DEGREES(numeric_exp)</i>	Returns the number of degrees converted from <i>numeric_exp</i> radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>EXP(double_exp)</i>	Returns the exponential value of <i>double_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>FLOOR(numeric_exp)</i>	Returns the largest integer less than or equal to <i>numeric_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>LOG(double_exp)</i>	Returns the natural logarithm of <i>double_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>LOG10(double_exp)</i>	Returns the base 10 logarithm of <i>double_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>MOD(integer_exp1, integer_exp2)</i>	Returns the remainder (modulus) of <i>integer_exp1</i> divided by <i>integer_exp2</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

Table 17. Numeric scalar functions (continued)

Numeric scalar function	Description	Servers that support the function
<i>PI()</i>	Returns the constant value of pi as a floating point value.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>POWER(numeric_exp, integer_exp)</i>	Returns the value of <i>numeric_exp</i> to the power of <i>integer_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>RADIANS(numeric_exp)</i>	Returns the number of radians converted from <i>numeric_exp</i> degrees.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>RAND([integer_exp])</i>	Returns a random floating point value using <i>integer_exp</i> as the optional seed value.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>ROUND(numeric_exp, integer_exp.)</i>	Returns <i>numeric_exp</i> rounded to <i>integer_exp</i> places right of the decimal point. If <i>integer_exp</i> is negative, <i>numeric_exp</i> is rounded to $ integer_exp $ places to the left of the decimal point.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>SIGN(numeric_exp)</i>	Returns an indicator or the sign of <i>numeric_exp</i> . If <i>numeric_exp</i> is less than zero, -1 is returned. If <i>numeric_exp</i> equals zero, 0 is returned. If <i>numeric_exp</i> is greater than zero, 1 is returned.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>SIN(double_exp)</i>	Returns the sine of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>SQRT(double_exp)</i>	Returns the square root of <i>double_exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>TAN(double_exp)</i>	Returns the tangent of <i>double_exp</i> , where <i>double_exp</i> is an angle expressed in radians.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>TRUNCATE(numeric_exp, integer_exp)</i>	Returns <i>numeric_exp</i> truncated to <i>integer_exp</i> places right of the decimal point. If <i>integer_exp</i> is negative, <i>numeric_exp</i> is truncated to $ integer_exp $ places to the left of the decimal point.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i

Date and time functions

The date and time functions in this section are supported by 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 18. Date and time scalar functions

Date and time scalar function	Description	Servers that support the function
<i>CURDATE()</i>	Returns the current date as a date value.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>CURTIME()</i>	Returns the current local time as a time value.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>DAYNAME(date_exp)</i>	Returns a character string containing the name of the day (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday) for the day portion of <i>date_exp</i> .	DB2 for Linux, UNIX, and Windows
<i>DAYOFMONTH (date_exp)</i>	Returns the day of the month in <i>date_exp</i> as an integer value in the range of 1-31.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>DAYOFWEEK(date_exp)</i>	Returns the day of the week in <i>date_exp</i> as an integer value in the range 1-7, where 1 represents Sunday.	DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i
<i>DAYOFWEEK_ISO(date_exp)</i>	Returns the day of the week in <i>date_exp</i> as an integer value in the range 1-7, where 1 represents Monday. Note the difference between this function and the <i>DAYOFWEEK()</i> function, where 1 represents Sunday.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, IBM DB2 for IBM i
<i>DAYOFYEAR(date_exp)</i>	Returns the day of the year in <i>date_exp</i> as an integer value in the range 1-366.	DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i

Table 18. Date and time scalar functions (continued)

Date and time scalar function	Description	Servers that support the function
<i>HOUR(time_exp)</i>	Returns the hour in <i>time_exp</i> as an integer value in the range of 0-23.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>JULIAN_DAY(date_exp)</i>	Returns the number of days between <i>date_exp</i> and January 1, 4712 B.C. (the start of the Julian date calendar).	DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i
<i>MINUTE(time_exp)</i>	Returns the minute in <i>time_exp</i> as integer value in the range of 0-59.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>MONTH(date_exp)</i>	Returns the month in <i>date_exp</i> as an integer value in the range of 1-12.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>MONTHNAME(date_exp)</i>	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 <i>date_exp</i> .	DB2 for Linux, UNIX, and Windows
<i>NOW()</i>	Returns the current date and time as a timestamp value.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>QUARTER(date_exp)</i>	Returns the quarter in <i>date_exp</i> as an integer value in the range of 1-4.	DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i
<i>SECOND(time_exp)</i>	Returns the second in <i>time_exp</i> as an integer value in the range of 0-59.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>SECONDS_SINCE_MIDNIGHT(time_exp)</i>	Returns the number of seconds in <i>time_exp</i> relative to midnight as an integer value in the range of 0-86400. If <i>time_exp</i> includes a fractional seconds component, the fractional seconds component will be discarded.	DB2 for Linux, UNIX, and Windows

Table 18. Date and time scalar functions (continued)

Date and time scalar function	Description	Servers that support the function
<p><i>TIMESTAMPADD(interval, integer_exp, timestamp_exp)</i></p>	<p>Returns the timestamp calculated by adding <i>integer_exp</i> intervals of type <i>interval</i> to <i>timestamp_exp</i>. Valid values of interval are:</p> <ul style="list-style-type: none"> • 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 <p>where fractional seconds are expressed in 1/1000000000 second. If <i>timestamp_exp</i> specifies a time value and <i>interval</i> specifies days, weeks, months, quarters, or years, the date portion of <i>timestamp_exp</i> is set to the current date before calculating the resulting timestamp. If <i>timestamp_exp</i> is a date value and <i>interval</i> specifies fractional seconds, seconds, minutes, or hours, the time portion of <i>timestamp_exp</i> is set to 00:00:00.000000 before calculating the resulting timestamp. An application determines which intervals are supported by calling <i>SQLGetInfo()</i> with the SQL_TIMEDATE_ADD_INTERVALS option.</p>	<p>DB2 for Linux, UNIX, and Windows</p>
<p><i>TIMESTAMPDIFF(interval, timestamp_exp1, timestamp_exp2)</i></p>	<p>Returns the integer number of intervals of type <i>interval</i> by which <i>timestamp_exp2</i> is greater than <i>timestamp_exp1</i>. Valid values of interval are:</p> <ul style="list-style-type: none"> • 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 <p>where fractional seconds are expressed in 1/1000000000 second. If either timestamp expression is a time value and <i>interval</i> 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 <i>interval</i> 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 <i>SQLGetInfo()</i> with the SQL_TIMEDATE_DIFF_INTERVALS option.</p>	<p>DB2 for Linux, UNIX, and Windows</p>
<p><i>WEEK(date_exp)</i></p>	<p>Returns the week of the year in <i>date_exp</i> as an integer value in the range of 1-54.</p>	<p>DB2 for Linux, UNIX, and Windows, IBM DB2 for IBM i</p>

Table 18. Date and time scalar functions (continued)

Date and time scalar function	Description	Servers that support the function
<i>WEEK_ISO(date_exp)</i>	<p>Returns the week of the year in <i>date_exp</i> 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.</p> <p>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.</p>	DB2 for Linux, UNIX, and Windows
<i>YEAR(date_exp)</i>	Returns the year in <i>date_exp</i> as an integer value in the range of 1-9999.	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

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 before Version 7, these functions might 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 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 19. System scalar functions

System scalar function	Description	Servers that support the function
<i>DATABASE()</i>	Returns the name of the database corresponding to the connection handle (<i>hdbc</i>). (The name of the database is also available via <i>SQLGetInfo()</i> by specifying the information type SQL_DATABASE_NAME.)	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

Table 19. System scalar functions (continued)

System scalar function	Description	Servers that support the function
<i>IFNULL(exp, value)</i>	If <i>exp</i> is null, <i>value</i> is returned. If <i>exp</i> is not null, <i>exp</i> is returned. The possible data type(s) of <i>value</i> must be compatible with the data type of <i>exp</i> .	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i
<i>USER()</i>	Returns the user's authorization name. (The user's authorization name is also available via <i>SQLGetInfo()</i> by specifying the information type <i>SQL_USER_NAME</i> .)	DB2 for Linux, UNIX, and Windows, DB2 for z/OS, DB2 Server for VM and VSE, IBM DB2 for IBM i

Conversion function

The conversion function is supported by 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 20. Conversion Function

Conversion scalar function	Description	Servers that support the function
<i>CONVERT(expr_value, data_type)</i>	<ul style="list-style-type: none"> <i>data_type</i> indicates the data type of the converted representation of <i>expr_value</i>, and can be either <i>SQL_CHAR</i> or <i>SQL_DOUBLE</i>. <i>expr_value</i> 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 <i>SQLGetInfo()</i> with the appropriate convert function masks to determine which conversions are supported by the data source. 	DB2 for Linux, UNIX, and Windows

Chapter 17. Non-Java client support for high availability on IBM data servers

Client applications that connect to DB2 for Linux, UNIX, and Windows, DB2 for z/OS, or IBM Informix can easily take advantage of the high availability features of those data servers.

Client applications can use the following high availability features:

- Automatic client reroute

Automatic client reroute capability is available on all IBM data servers.

Automatic client reroute uses information that is provided by the data servers to redirect client applications from a server that experiences an outage to an alternate server. Automatic client reroute enables applications to continue their work with minimal interruption. Redirection of work to an alternate server is called *failover*.

For connections to DB2 for z/OS data servers, automatic client reroute is part of the workload balancing feature. In general, for DB2 for z/OS, automatic client reroute should not be enabled without workload balancing.

- Client affinities

Client affinities is a failover solution that is controlled completely by the client. It is intended for situations in which you need to connect to a particular primary server. If an outage occurs during the connection to the primary server, you use client affinities to enforce a specific order for failover to alternate servers.

Client affinities is not applicable to a DB2 for z/OS data sharing environment, because all members of a data sharing group can access data concurrently. Data sharing is the recommended solution for high availability for DB2 for z/OS.

- Workload balancing

Workload balancing is available on all IBM data servers. Workload balancing ensures that work is distributed efficiently among servers in an IBM Informix high-availability cluster, DB2 for z/OS data sharing group, or DB2 for Linux, UNIX, and Windows DB2 pureScale® instance.

The following table provides links to server-side information about these features.

Table 21. Server-side information about high availability

Data server	Related topics
DB2 for Linux, UNIX, and Windows	<ul style="list-style-type: none">• DB2 pureScale: DB2 pureScale Feature roadmap documentation• Automatic client reroute: Automatic client reroute roadmap
IBM Informix	Manage Cluster Connections with the Connection Manager
DB2 for z/OS	Communicating with data sharing groups

Important: For connections to DB2 for z/OS, this information discusses direct connections to DB2 for z/OS. For information about high availability for connections through DB2 Connect server, see the DB2 Connect documentation.

Non-Java client support for high availability for connections to DB2 for Linux, UNIX, and Windows

DB2 for Linux, UNIX, and Windows servers provide high availability for client applications, through workload balancing and automatic client reroute. This support is available for applications that use non-Java clients (ODBC, CLI, .NET, OLE DB, PHP, Ruby, or embedded SQL), as well as Java™ clients (JDBC, SQLJ, or pureQuery).

For non-Java clients, you must use one of the listed clients or client packages to take advantage of high availability support:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

High availability support for connections to DB2 for Linux, UNIX, and Windows servers includes:

Automatic client reroute

This support enables a client to recover from a failure by attempting to reconnect to the database through an alternate server. Reconnection to another server is called *failover*. Client support for automatic client reroute is enabled by default for non-Java clients that connect to DB2 for Linux, UNIX, and Windows.

Servers can provide automatic client reroute capability in any of the succeeding ways:

- Several servers are configured in a DB2 pureScale instance. A connection to a database is a connection to a member of that DB2 pureScale instance. Failover involves reconnection to another member of the DB2 pureScale instance. This environment requires that clients use TCP/IP to connect to the DB2 pureScale instance.
- A DB2 pureScale instance and an alternate server that is defined for a database. Failover first involves reconnection to another member of the DB2 pureScale instance. Failover to the server is attempted only if no member of the DB2 pureScale instance is available.
- A DB2 pureScale instance is defined for the primary server, and another DB2 pureScale instance is defined for the server. Failover first involves reconnection to another member of the primary DB2 pureScale instance. Failover to the alternate DB2 pureScale instance is attempted only if no member of the primary DB2 pureScale instance is available.
- A database is defined on a single server. The configuration for that database includes specification of an alternate server. Failover involves reconnection to the alternate server.

Alternate groups are an additional failover mechanism for automatic client rerouting when connectivity to the current group cannot be re-established. A *group* is a database that is created in a DB2 instance. In DB2 pureScale or partitioned database environments, all the participant database servers for a database are also considered a group. The database to which your application explicitly connects to is called the *primary group*.

For CLI or .NET client applications, failover for automatic client reroute can be *seamless* or *non-seamless*. With non-seamless failover, when the client application

reconnects to another server an error is always returned to the application to indicate that failover (connection to the alternate server) occurred. With seamless failover, the client does not return an error if a connection failure and successful reconnection to an alternate server or alternate group occurs during the execution of the first SQL statement in a transaction.

In a DB2 pureScale instance, you can use automatic client reroute support without workload balancing or with workload balancing.

Workload balancing

Workload balancing can improve the availability of a DB2 pureScale instance.

With workload balancing, a DB2 pureScale instance ensures that work is distributed efficiently among members.

Non-Java clients on any operating system support workload balancing. The connection from the client to the DB2 pureScale instance must use TCP/IP.

When workload balancing is enabled, the client gets frequent status information about the members of the DB2 pureScale instance through a server list. The client caches the server list and uses the information in it to determine the member to which the next transaction should be routed.

For non-Java clients, the server list is cached in the application process. It is shared for workload balancing only by connections in that process.

DB2 for Linux, UNIX, and Windows supports two types of workload balancing:

Connection-level workload balancing

Connection-level workload balancing is performed at connection boundaries. It is supported only by non-Java clients. Client support for connection-level workload balancing is enabled by default for non-Java clients that connect to DB2 for Linux, UNIX, and Windows.

Connection-level load balancing is most effective for connections whose duration is short.

Transaction-level workload balancing

Transaction-level workload balancing is performed at transaction boundaries. Client support for transaction-level workload balancing is disabled by default for clients that connect to DB2 for Linux, UNIX, and Windows.

Transaction-level load balancing is most effective for connections whose duration is long.

Client affinities

Client affinities is an automatic client reroute solution that is controlled completely by the client. It is intended for situations in which you must connect to a particular primary server. If an outage occurs during the connection to the primary server, you use client affinities to enforce a specific order for failover to alternate servers.

Configuration of DB2 for Linux, UNIX, and Windows automatic client reroute support for non-Java clients

For connections to DB2 for Linux, UNIX, and Windows databases, the process for configuration of automatic client reroute support on non-Java clients is the same for a non-DB2 pureScale environment and a DB2 pureScale environment.

Automatic client reroute capability is enabled at a non-Java client by default. You must connect to the primary server in a non-DB2 pureScale environment or to a DB2 pureScale instance in a DB2 pureScale environment.

At the first successful connection to the server, the client obtains a list, which the client stores in memory, from the server of all the available alternate servers. If the first connection fails, the client looks for a list of alternate servers that are defined in the `db2dsdriver.cfg` file, in the `<ar>` section, under the `<aternateserverlist>` tag.

If the `db2dsdriver.cfg` file has no alternate servers that are defined in the `<acr>` section, upon the first successful connection to the server, the client creates a local cache file, `srvr1st.xml`. The client updates the file with the server's list of available alternate servers. This file is refreshed whenever a new connection is made and the server's list differs from the contents of the client `srvr1st.xml` file.

When a client uses the `srvr1st.xml` file to locate an alternate server, it writes a record to the `db2diag` log files. You can monitor this log to determine how frequently initial sever connections are failing.

The table 1 describes the basic settings to establish a connection for non-Java applications.

Table 22. Basic settings to establish a connection to a DB2 for Linux, UNIX, and Windows database in non-Java applications

Client setting	Value for a non-DB2 pureScale environment	Value for a DB2 pureScale environment
Connection address:		
Database host ¹	The IP address of the primary server.	The IP address of a member of a DB2 pureScale instance. ²
Database port ¹	The SQL port number of the primary server.	The SQL port number of a member of a DB2 pureScale instance. ²
Database name ¹	The database name.	The database name.

Table 22. Basic settings to establish a connection to a DB2 for Linux, UNIX, and Windows database in non-Java applications (continued)

Client setting	Value for a non-DB2 pureScale environment	Value for a DB2 pureScale environment
Note:		
1. Depending on the client that you use, connection information is defined in one of several possible sources:		
<ul style="list-style-type: none"> • If you are using one of the data server drivers, a CLI, or an open source application that uses IBM Data Server Client or IBM Data Server Runtime Client: <ul style="list-style-type: none"> – If host, port, and database information is provided in the connection string in an application, the DB2 database system uses that information. – If host, port, and database information is not provided in the connection string in an application, the driver uses a <code>db2cli.ini</code> file, and this information is provided in the <code>db2cli.ini</code> file. – If host, port, and database information is not provided in the connection string in the application or the <code>db2cli.ini</code> file, the DB2 database system uses information in the <code>db2dsdriver</code> configuration file. • If you are using a .NET application with the IBM Data Server Client or the IBM Data Server Runtime Client, connection information can be defined in the database catalog, connection string, <code>db2dsdriver</code> configuration file, or .NET object properties. • If you are using an embedded SQL application with the IBM Data Server Client or the IBM Data Server Runtime Client, connection information can be defined in the database catalog, connection string, or <code>db2dsdriver</code> configuration file. 		
2. Alternatively, you can use a distributor, such as WebSphere Application Server Network Deployment, or multihomed DNS to establish the initial connection to the database.		
<ul style="list-style-type: none"> • For a distributor, you specify the IP address and port number of the distributor. The distributor analyzes the current workload distribution, and uses that information to forward the connection request to one of the members of the DB2 pureScale instance. • For multihomed DNS, you specify an IP address and port number that can resolve to the IP address and port number of any member of the DB2 pureScale instance. Multihomed DNS processing selects a member based on criteria, such as simple round-robin selection or member workload distribution. 		

You can set configuration keywords or registry variables in the `db2dsdriver.cfg` file to refine automatic client reroute behavior. You can use the configuration keywords in the table 2 to control automatic client reroute. The keywords are described for the case in which client affinities are not enabled.

In case of changes to the `db2dsdriver.cfg` file, your CLI application can start the `SQLReloadConfig` function to validate the entries for all alternate servers within the `<acr>` section.

Table 23. Settings to control automatic client reroute behavior

Element in the <code><acr></code> section of the <code>db2dsdriver</code> configuration file	Value
<code>enableAcr</code> parameter	Specifies whether automatic client reroute is in effect. The default is true.
<code>enableSeamlessAcr</code> parameter	Specifies whether seamless failover can occur. If <code>enableAcr</code> is set to true, the default for <code>enableSeamlessAcr</code> is true. <code>enableSeamlessACR</code> applies only to the members within a group or cluster.

Table 23. Settings to control automatic client reroute behavior (continued)

Element in the <acr> section of the db2dsdriver configuration file	Value
enableAlternateGroupSeamlessACR parameter	Specifies seamless or non-seamless failover behavior across groups. The default is false. You must define this parameter in the <alternategroup> element in the <acr> section. To set this parameter to true, you must also set enableSeamlessACR to true. Setting this parameter to true, does not affect the setting of enableSeamlessACR. If a successful connection is established to a server in the <i>alternategroup</i> section, the rules for seamless or non-seamless behavior still apply.
acrRetryInterval parameter	The number of seconds to wait between consecutive connection retries. The registry variable DB2_CONNRETRIES_INTERVAL overrides this value. The valid range is 0 to MAX_INT. The default is no wait (0), if DB2_CONNRETRIES_INTERVAL is not set.
maxAcrRetries parameter	Specifies the maximum number of connection attempts for automatic client reroute. The valid range is 0 to MAX_INT. The value of the DB2_MAX_CLIENT_CONNRETRIES registry variable overrides this value. If you do not set the DB2_MAX_CLIENT_CONNRETRIES registry variable or maxAcrRetries parameter, by default the connection is tried again for 10 minutes. However, if alternate groups are defined, by default, a connection is attempted for 2 minutes. Setting this value to 0 disables ACR.
enableAlternateServerListFirstConnect parameter	Specifies whether there is an alternate server list that is used only if a failure occurs on the first connection to the data server. The default is false. When the value of enableAlternateServerListFirstConnect is true, automatic client reroute with seamless failover is implicitly enabled, regardless of the other settings that are specified for automatic client reroute in the db2dsdriver configuration file. To use this feature, you also require an <alternateserverlist> element in the db2dsdriver configuration file.
alternateserverlist parameter	Specifies a set of server names and port numbers that identify alternate servers to which a connection is attempted if a failure occurs on the first connection to the database. The alternate server list is not used after the first connection. In a DB2 pureScale environment, the entries in the list can be members of a DB2 pureScale instance. In a non-DB2 pureScale environment, there is an entry for the primary server and an entry for the high availability disaster recovery (HADR) standby server. The alternate server list is not used after the first connection.

The registry variables in table 3 controls retry behavior of automatic client reroute.

Table 24. Registry variables to control automatic client reroute retry behavior

Registry variable	Value
DB2_MAX_CLIENT_CONNRETRIES	The maximum number of connection retries for automatic client reroute. The default is 30 if DB2_CONNRETRIES_INTERVAL is set.
DB2_CONNRETRIES_INTERVAL	The number of seconds between consecutive connection retries. The default is 10 if DB2_MAX_CLIENT_CONNRETRIES is set.

If neither registry variable is set, and **maxAcrRetries** and **acrRetryInterval** are also not set, automatic client reroute processing tries the connection to a database again for up to 10 minutes, with no wait between retries.

For embedded SQL, CLI, OLE DB, and ADO.NET applications, you can set a connection timeout value. That value specifies the number of seconds that the client application waits for a connection to a database to be established. You must set the connection timeout value to a value that is equal to or greater than the maximum time that it takes to connect to the server. Otherwise, the connection might time out and be rerouted to the alternate server by client reroute. For example, if on a normal day it takes about 10 seconds to connect to the server, and on a busy day it takes about 20 seconds, you should set the connection timeout value t to at least 20 seconds.

Example of enabling DB2 for Linux, UNIX, and Windows automatic client reroute support in non-Java clients

You can fine-tune non-Java client setup for DB2 for Linux, UNIX, and Windows automatic client reroute (acr) support by setting values for several keywords in the `db2dsdriver.cfg` configuration file.

Note that if you do not define a list of alternate servers in the `db2dsdriver.cfg` configuration file, at the first successful connection to the server, the client obtains from the server a list of all available alternate servers. The client stores the list in memory and also creates a local cache file, `svr1st.xml` that contains the server's list of alternate servers. This file is refreshed whenever a new connection is made and the server's list differs from the contents of the client `svr1st.xml` file.

Suppose that database sample is a DB2 pureScale instance with two members, at server `db2luwa` and port 446, and server `db2luwb` and port 446. The database has alternate server `db2luwc` and port 446 defined.

You want to fine-tune default automatic client reroute support by modifying these items:

Automatic client reroute characteristic	db2dsdriver.cfg configuration keyword	Desired value
Number of times to retry the connection to the alternate server	<code>maxAcrRetries</code>	10
Number of seconds to wait between retries	<code>acrRetryInterval</code>	5

Automatic client reroute characteristic	db2dsdriver.cfg configuration keyword	Desired value
Whether to try another server if the initial connection to the database fails	enableAlternateServerListFirstConnect	true
The host names and port numbers of the servers to try if the initial connection to the database fails	<alternateserverlist> Note that if you do not define a list of alternate servers in the db2dsdriver.cfg configuration file, at the first successful connection to the server, the client obtains from the server a list of all available alternate servers. The client stores the list in memory and also creates a local cache file, srvr1st.xml that contains the server's list of alternate servers. This file is refreshed whenever a new connection is made and the server's list differs from the contents of the client srvr1st.xml file. The srvr1st.xml file is located under CLIENT_CONFIG_DIR in the cfgcache directory.	Host names and port numbers: <ul style="list-style-type: none"> • db2luwa.luw.ibm.com, 446 • db2luwb.luw.ibm.com, 446 • db2luwc.luw.ibm.com, 446

Use the following db2dsdriver.cfg configuration file to implement these changes to automatic reroute behavior:

```

<configuration>
  <dsncollection>
    <dsn alias="sample" name="sample" host="db2luw.luw.ibm.com" port="446">
    </dsn>
  </dsncollection>
  <databases>
    <database name="sample" host="db2luw.luw.ibm.com" port="446">
      <acr>
        <parameter name="enableAcr" value="true">
        </parameter>
        <parameter name=
        <parameter name="maxAcrRetries" value="10">
        </parameter>
        <parameter name="acrRetryInterval" value="5">
        </parameter>
        <parameter name="enableAlternateServerListFirstConnect"
        value="true"></parameter>
      <alternateserverlist>
        <server name="server1" hostname="db2luwa.luw.ibm.com" port="446">
        </server>
        <server name="server2" hostname="db2luwb.luw.ibm.com" port="446">
        </server>
        <server name="server3" hostname="db2luwc.luw.ibm.com" port="446">
        </server>
      </alternateserverlist>
    </acr>
    </database>
  </databases>
</configuration>

```

Configuration of DB2 for Linux, UNIX, and Windows workload balancing support for non-Java clients

For connections to DB2 for Linux, UNIX, and Windows data servers in a DB2 pureScale instance, connection-level workload balancing is enabled at a non-Java client by default. Transaction-level workload balancing capability must be enabled explicitly.

The following table describes the basic settings to enable connection-level workload balancing support for non-Java applications.

Table 25. Basic settings to enable DB2 for Linux, UNIX, and Windows connection-level workload balancing support in non-Java applications

Client setting	Value
Connection address:	
database host ¹	The IP address of a member of a DB2 pureScale instance. ²
database port ¹	The SQL port number of a member of a DB2 pureScale instance ²
database name ¹	The database name

Note:

1. Depending on the client that you use, connection information is defined in one of several possible sources:
 - If you are using one of the data server drivers or a CLI or open source application that uses IBM Data Server Client or IBM Data Server Runtime Client:
 - If host, port, and database information is provided in a connection string in an application, DB2 uses that information.
 - If host, port, and database information is not provided in the connection string in an application, the driver uses a `db2cli.ini` file, and this information is provided in the `db2cli.ini` file, DB2 uses that information.
 - If host, port, and database information is not provided in the connection string in the application or the `db2cli.ini` file, the DB2 driver uses the information in the `db2dsdriver.cfg` configuration file.
 - If you are using a .NET application or an application that uses embedded SQL with the IBM Data Server Client or the IBM Data Server Runtime Client, connection information comes a source that is not the `db2dsdriver.cfg` configuration file. Possible sources include the database catalog, connection string, `db2cli.ini` file, or .NET object properties.
2. Alternatively, you can use a distributor, such as Websphere Application Server Network Deployment, or multihomed DNS to establish the initial connection to the database.
 - For a distributor, you specify the IP address and port number of the distributor. The distributor analyzes the current workload distribution, and uses that information to forward the connection request to one of the members of the DB2 pureScale instance.
 - For multihomed DNS, you specify an IP address and port number that can resolve to the IP address and port number of any member of the DB2 pureScale instance. Multihomed DNS processing selects a member based on some criterion, such as simple round-robin selection or member workload distribution.

The following configuration keyword in the `db2dsdriver.cfg` file can be used to change the connection-level workload balancing setting.

Table 26. Setting to control connection-level workload balancing behavior

Element in the db2dsdriver.cfg configuration file	Section	Value
connectionLevelLoadBalancing parameter	<database>	Specifies whether connection-level load balancing is in effect. It is true by default.

You can use the following configuration keywords in the db2dsdriver.cfg file to enable and fine-tune transaction-level workload balancing.

Table 27. Settings to control transaction-level workload balancing behavior

Element in the db2dsdriver.cfg configuration file	Section	Value
connectionLevelLoadBalancing parameter	<database>	Must be set to true if you want to use transaction-level workload balancing. The setting is true by default unless the server accessed is a DB2 for z/OS server; in that case, the default is false.
enableWLB parameter	<wlb>	Specifies whether transaction-level workload balancing is in effect. It is false by default.
maxTransportIdleTime	<wlb>	Specifies the maximum elapsed time in number of seconds before an idle transport is dropped. The default is 60 . The minimum value is 0.
maxTransportWaitTime	<wlb>	Specifies the number of seconds that the client waits for a transport to become available. The default is 1. Specifying a value as -1 means unlimited wait time. The minimum supported value is 0.
maxTransports	<wlb>	Specifies the maximum number of physical connections can be made for each application process that connects to the DB2 pureScale instance. The default is -1 which means establish as many as needed.
maxRefreshInterval	<wlb>	Specifies the maximum elapsed time in number of seconds before the server list is refreshed. The default is 10. The minimum supported value is 0.

Example of enabling DB2 for Linux, UNIX, and Windows workload balancing support in non-Java clients

DB2 for Linux, UNIX, and Windows workload balancing requires a DB2 pureScale environment. Before you can use DB2 for Linux, UNIX, and Windows workload balancing support in CLI, .NET, or embedded SQL applications, you need to update the db2dsdriver.cfg configuration file with the appropriate settings, and connect to a member of the DB2 pureScale environment.

The following example demonstrates setting up a CLI client to take advantage of DB2 for Linux, UNIX, and Windows workload balancing support.

Before you can set up the client, you need to configure a DB2 pureScale instance.

These steps demonstrate client setup:

1. Create a `db2dsdriver.cfg` file that enables transaction-level workload balancing. In this example:

- If the first connection to the database fails, the connection needs to be tried on alternate servers.

Note that if you do not define a list of alternate servers in the `db2dsdriver.cfg` configuration file, at the first successful connection to the server, the client obtains from the server a list of all available alternate servers. The client stores the list in memory and also creates a local cache file, `svr1st.xml` that contains the server's list of alternate servers. This file is refreshed whenever a new connection is made and the server's list differs from the contents of the client `svr1st.xml` file.

- For transaction-level workload balancing for this database, the maximum number of physical connections needs to be 80.
- Connections can use the defaults for all other transaction-level workload balancing parameters.

The `db2dsdriver.cfg` file looks like this:

```
<configuration>
  <dsncollection>
    <dsn alias="LUWDS1" name="LUWDS1" host="luw1ds.toronto.ibm.com"
      port="50000">
    </dsn>
  </dsncollection>
  <databases>
    <!-- In this example, the host and port represent a member of a
      DB2 pureScale instance -->
    <database name="LUWDS1" host="luw.ds1.ibm.com" port="50000">
      <!-- database-specific parameters -->
      <wlb>
        <!-- Enable transaction-level workload balancing -->
        <parameter name="enableWLB" value="true" />
        <!-- maxTransports represents the maximum number of physical
          connections -->
        <parameter name="maxTransports" value="80" />
      </wlb>
      <acr>
        <!-- acr is already enabled by default -->
        <!-- Enable server list for application first connect -->
        <parameter name="enableAlternateServerListFirstConnect"
          value="true" />
        <alternateserverlist>
          <!-- Alternate server 1 -->
          <parameter name="server" value="luw2ds.toronto.ibm.com" />
          <parameter name="port" value="50001" />
          <!-- Alternate server 2 -->
          <parameter name="server" value="luw3ds.toronto.ibm.com" />
          <parameter name="port" value="50002" />
          <!-- Alternate server 3 -->
          <parameter name="server" value="luw4ds.toronto.ibm.com" />
          <parameter name="port" value="50003" />
        </alternateserverlist>
      </acr>
    </database>
  </databases>
</configuration>
```

2. Suppose that the database name `LUWDS1` represents a DB2 pureScale instance. In a CLI application, use code like this to connect to the DB2 pureScale instance:

```
...
SQLHDBC      hDbc    = SQL_NULL_HDBC;
SQLRETURN    rc      = SQL_SUCCESS;
SQLINTEGER   RETCODE = 0;
```

```

char          *ConnStrIn =
              "DSN=LUWDS1;PWD=mypass";
              /* dsn matches the database name in the configuration file */
char          ConnStrOut [200];
SQLSMALLINT  cbConnStrOut;
int          i;
char         *token;

...
/*****
/* Invoke SQLDriverConnect
*****/
RETCODE = SQLDriverConnect (hDbc
                          ,
                          NULL
                          ,
                          (SQLCHAR *)ConnStrIn ,
                          strlen(ConnStrIn) ,
                          (SQLCHAR *)ConnStrOut,
                          sizeof(ConnStrOut) ,
                          &cbConnStrOut
                          ,
                          SQL_DRIVER_NOPROMPT);

...

```

Operation of the automatic client reroute feature when connecting to the DB2 for Linux, UNIX, and Windows server from an application other than a Java application

Automatic client reroute provides failover support when an IBM data server client loses connectivity to the primary server for a DB2 for Linux, UNIX, and Windows database. Automatic client reroute enables the client to recover from a failure by attempting to reconnect to the database through an alternate server.

If automatic client reroute is enabled for a connection to a database, the following process typically occurs when a client encounters a connection failure with an existing connection:

1. The client attempts to execute an SQL statement by using an existing connection and encounters a failure.
2. The client uses the server list that is returned after the last successful connection to identify which server to access, and attempts to reconnect to the database.

In a non-DB2 pureScale environment, the server list contains two entries: one for the primary server and one for the alternate server.

In a DB2 pureScale environment, the server list contains an entry for each member of the DB2 pureScale instance. In addition, if an alternate server is defined for the database, the server list also contains an entry for that alternate server. An entry for a member of the DB2 pureScale instance includes capacity information. If connection-level workload balancing is enabled at the client, the client uses that information to connect to the server with the highest unused capacity. The entry for an alternate server has no capacity information. A connection to the alternate server is attempted only if connections to all of the DB2 pureScale members fail.

3. If the automatic client reroute process can reconnect the application to the database, the client reconstructs the execution environment for the newly established connection. The client receives an updated copy of the server list with updated server information. The error SQL30108N is returned to the application to indicate that the failed database connection has been recovered and that the transaction has been rolled back. The application is then responsible for further recovery, including repeating any work that was rolled back.

If the SQL statement that fails is the first SQL statement in the transaction, automatic client reroute with seamless failover is enabled, and the client is CLI or .NET, the driver replays the failed SQL operation as part of automatic client reroute processing. If the connection is successful, no error is reported to the application, and the transaction is not rolled back. The connectivity failure and subsequent recovery are hidden from the application.

4. If automatic client reroute is unable to reconnect to the database, the error SQL30081N is returned to the application. The application is then responsible for recovering from the connection failure (for example, by attempting to connect to the database by itself).

Automatic client reroute is also used when a client encounters a connection failure with a new connection. In this case, however, if reconnection is successful, no error is returned to the application to indicate that the failed database connection has been recovered. If reconnection fails, the error SQL30081N is returned.

If all the data, including the end of file (EOF) character, is returned in the first query block or in a subsequent fetch request, the CLI driver can perform seamless failover when you issue a COMMIT or ROLLBACK statement after the server becomes unreachable. For seamless failover to take place, the following conditions must be met:

- You must enable both the **enableAcr** and **enableSeamlessAcr** parameters.
- The cursor must have blocking enabled.
- The cursor must be either read only or forward only.

If the application turned off the automatic closing of cursors at the server side by setting the **SQL_ATTR_EARLYCLOSE** attribute to OFF, the CLI driver can perform seamless failover only if the application processed the entire result set.

You can configure the **FetchBufferSize** keyword to ensure that the size of the result set that the CLI driver prefetches is sufficient to include the EOF character in the first query block. See the Related reference section for further details on **FetchBufferSize**.

The non-seamless ACR environments have one of the following settings:

- The **enableAcr** keyword is set to TRUE but the **enableSeamlessAcr** is set to FALSE.
- The **enableACR** keyword and the **enableSeamlessAcr** keyword are set to TRUE but the seamless ACR environment cannot be enabled due to following reasons:
 - Session resources are created in the transaction.
 - The failed SQL statement was not the first SQL statement of the transaction.

In the non-seamless ACR environment, the **ReceiveTimeout** event triggers the ACR connection error (SQL30108N).

If the **QueryTimeout** parameter is set in the non-seamless ACR environment, the following behaviors are observed:

- If the connection failure occurs before the query timeout event, the ACR connection error (SQL30108N with reason code 1, and failure code 1, 2 or 3) is returned to the application.
- If the **Interrupt** parameter is set to the default value of 1 and the query timeout event occurs, the SQL0952N error is returned to the application.
- If the **Interrupt** parameter is set to the value of 2 and the query timeout event occurs, the ACR connection error (SQL30108N with failure code 4 and error code 1) is returned to the application.

When the **Interrupt** parameter is set to the value of 2 and the `SQLCancel()` API is explicitly sent from the application while SQL statement is being executed, the ACR connection error (SQL30108N failure code 4 and error code 2) is returned to the application.

If the **tcPIPConnectTimeout** parameter and the **memberConnectTimeout** parameter is set and the TCPIP Connection timeout or the member connection timeout event occurs in the non-seamless ACR environment, the CLI driver reconnects to a new available member but the ACR error (SQL30108N) is not returned to the application.

Operation of transaction-level workload balancing for connections to DB2 for Linux, UNIX, and Windows

Transaction-level workload balancing for connections to DB2 for Linux, UNIX, and Windows contributes to high availability by balancing work among servers in a DB2 pureScale instance at the start of a transaction.

The following overview describes the steps that occur when a client connects to a DB2 for Linux, UNIX, and Windows DB2 pureScale instance, and transaction-level workload balancing is enabled:

1. When the client first establishes a connection to the DB2 pureScale instance, the member to which the client connects returns a server list with the connection details (IP address, port, and weight) for the members of the DB2 pureScale instance.

The server list is cached by the client. The default lifespan of the cached server list is 30 seconds.

2. At the start of a new transaction, the client reads the cached server list to identify a server that has unused capacity, and looks in the transport pool for an idle transport that is tied to the under-utilized server. (An idle transport is a transport that has no associated connection object.)
 - If an idle transport is available, the client associates the connection object with the transport.
 - If, after a user-configurable timeout period (`db2.jcc.maxTransportObjectWaitTime` for a Java client or `maxTransportWaitTime` for a non-Java client), no idle transport is available in the transport pool and no new transport can be allocated because the transport pool has reached its limit, an error is returned to the application.
3. When the transaction runs, it accesses the server that is tied to the transport.
4. When the transaction ends, the client verifies with the server that transport reuse is still allowed for the connection object.
5. If transport reuse is allowed, the server returns a list of SET statements for special registers that apply to the execution environment for the connection object.

The client caches these statements, which it replays in order to reconstruct the execution environment when the connection object is associated with a new transport.

6. The connection object is then dissociated from the transport, if the client determines that it needs to do so.
7. The client copy of the server list is refreshed when a new connection is made, or every 30 seconds, or the user-configured interval.

8. When transaction-level workload balancing is required for a new transaction, the client uses the previously described process to associate the connection object with a transport.

Alternate groups for connections to DB2 Database for Linux, UNIX, and Windows from non-Java clients

To improve high availability for non-Java clients in Version 9.7 Fix Pack 5 or later fix pack releases, use alternate groups as an additional failover mechanism for automatic client rerouting when connectivity to the current group cannot be re-established.

By default, non-Java clients have the automatic client reroute (ACR) enabled. This capability provides automatic failover to alternate servers within the current group when connectivity to a server cannot be re-established.

In addition to this ACR capability, you can define *alternate groups* as failover targets when connectivity to the current group cannot be established. To define alternate groups for non-Java clients:

- Define a <database> element for each alternate group inside the <alternategroup> element in the <acr> section of the db2dsdriver.cfg file. Do not specify <parameter> elements inside the <database> element, parameter settings are inherited from the primary group. You can define multiple <database> elements inside the <alternategroup> element. The order of the <database> elements is the order that is used during failover.
- If you want to suppress error messages from failover connections to the alternate group, set the enableAlternateGroupSeamlessACR parameter to true in <alternategroup> element.

The default ACR retry time period is 10 minutes. When you define alternate groups, that time period is reduced to 2 minutes.

When a non-Java client is connected to an alternate group, all the connection settings and the parameter settings for the <database> element in the primary group are inherited by the connection to the database in the alternate group.

After a non-Java client is connected to a database in the alternate group, no failback to the primary group is provided. To connect to the primary group again, the application or client must be restarted.

Alternate groups are only supported for ACR and workload balancing. If client affinities is configured, alternate group definitions are ignored.

Examples

Here is an example of alternate group definitions in the db2dsdriver.cfg file:

```
<dsncollection>
  <dsn alias="mydsn2" name="mydb2" host="myserver2.ibm.com" port="5912">
    ...
</dsncollection>

<databases>
  <database name="mydb2" host="myserver2.ibm.com" port="5912">
    <parameter name="IsolationLevel" value="4"/>
    ...
  <wlb>
    <parameter name="enableWLB" value="true"/>
```

```

</wlb>
<acr>
  ... (ACR parameters definition)
  <alternateserverlist>
    <server name="server1" hostname="db21uwa.luw.ibm.com" port="5912">
    </server>
    <server name="server2" hostname="db21uwb.luw.ibm.com" port="5912">
    </server>
  </alternateserverlist>
  <alternategroup>
    <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
    <database name="mydb3" host="myserver3.ibm.com" port="5912">
    </database>
    <database name="mydb4" host="myserver4.ibm.com" port="5912">
    </database>
  </alternategroup> </acr>
</database>

<database name="mydb3" host="myserver3.ibm.com" port="5912">
<parameter name="IsolationLevel" value="2"/>
<acr>
  <parameter name="enableACR" value="true"/>
  <alternateserverlist>
    <server name="server4" hostname="db21uwd.luw.ibm.com" port="5912">
    </server>
  </alternateserverlist>
  <alternategroup>
    <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
    <database name="mydb5" host="myserver5.ibm.com" port="5912">
    </database>
  </alternategroup> </acr>
  ...
</database>
</databases>

```

The following example scenarios demonstrate how automatic client rerouting works for alternate groups. The details about ACR failover to the current group are not covered in these scenarios to focus on the alternate groups failover details. These scenarios use the `db2dsdriver.cfg` sample file that is described in the previous paragraph.

First connection to the primary group fails

After a non-Java client fails to connect to the primary group on its first attempt, automatic client reroute failover to alternate servers in the current group also fails. In this example, the client performs the following actions:

1. The client fails to connect to *mydb2*.
2. The client fails to connect to *server1*.
3. The client fails to connect to *server2*.
4. The client tries to connect to an alternate group listed in the `<alternategroup>` section of the `db2dsdriver.cfg` file in the order specified in this file:
 - a. The client fails to connect to *mydb3*.
 - b. The client successfully connects to *mydb4*.

After connecting to *mydb4*, the rules for seamless or non-seamless behavior still apply. If the client would not be able to connect to *mydb4*, it would receive the SQL30081N error message.

Subsequent connection or existing connection to the primary server fails

After a non-Java client loses its connection to *mydb2*, automatic client reroute failover to alternate servers in the current group also fails. In this example, the client performs the following actions:

1. The client fails to connect to *server1*.
2. The client fails to connect to *server2*.
3. The client tries to connect to an alternate group listed in the <alternategroup> section of the *db2dsdriver.cfg* file in the order specified in this file:
 - a. The client successfully connects to *mydb3*.

After connecting to *mydb3*, the rules for seamless or non-seamless behavior still apply.

Existing connection to an alternate group fails

A non-Java client fails to connect to *mydb2*, automatic client reroute failover to alternate servers in the current group also fails, and then it successfully connects to the *mydb3* alternate group.

After the client loses its connection to *mydb3*, it attempts to connect to *mydb4*. In this example, the client fails to connect to *mydb4*.

The client receives the SQL30081N error message. You must restart the client or the application to try connecting to the primary group again.

Application programming requirements for high availability for connecting to DB2 for Linux, UNIX, and Windows servers

Failover for automatic client reroute can be seamless or non-seamless. If failover for connections to DB2 for Linux, UNIX, and Windows is not seamless, you need to add code to account for the errors that are returned when failover occurs.

If failover is non-seamless, and a connection is reestablished with the server, SQLCODE -4498 (for Java clients) or SQL30108N (for non-Java clients) is returned to the application. All work that occurred within the current transaction is rolled back. In the application, you need to:

- Check the reason code that is returned with the error. Determine whether special register settings on the failing data sharing member are carried over to the new (failover) data sharing member. Reset any special register values that are not current.
- Execute all SQL operations that occurred during the previous transaction.

The following conditions must be satisfied for failover for connections to DB2 for Linux, UNIX, and Windows to be seamless:

- The application programming language is Java, CLI, or .NET.
- The connection is not in a transaction. That is, the failure occurs when the first SQL statement in the transaction is executed.
- If transaction-level load balancing is enabled, the data server allows transport reuse at the end of the previous transaction.
- All global session data is closed or dropped.
- There are no open, held cursors.
- If the application uses CLI, the application cannot perform actions that require the driver to maintain a history of previously called APIs in order to replay the SQL statement. Examples of such actions are specifying data at execution time, performing compound SQL, or using array input.
- The application is not a stored procedure.
- Autocommit is not enabled. Seamless failover can occur when autocommit is enabled. However, the following situation can cause problems: Suppose that SQL work is successfully executed and committed at the data server, but the

connection or server goes down before acknowledgment of the commit operation is sent back to the client. When the client re-establishes the connection, it replays the previously committed SQL statement. The result is that the SQL statement is executed twice. To avoid this situation, turn autocommit off when you enable seamless failover.

Client affinities for clients that connect to DB2 for Linux, UNIX, and Windows

Client affinities is a client-only method for providing automatic client reroute capability.

Client affinities is available for applications that use CLI, .NET, or Java (IBM Data Server Driver for JDBC and SQLJ type 4 connectivity). All rerouting is controlled by the driver.

Client affinities is intended for situations in which you need to connect to a particular primary server. If an outage occurs during the connection to the primary server, you need to enforce a specific order for failover to alternate servers. You should use client affinities for automatic client reroute only if automatic client reroute that uses server failover capabilities does not work in your environment.

As part of configuration of client affinities, you specify a list of alternate servers, and the order in which connections to the alternate servers are tried. When client affinities is in use, connections are established based on the list of alternate servers instead of the host name and port number that are specified by the application. For example, if an application specifies that a connection is made to server1, but the configuration process specifies that servers should be tried in the order (server2, server3, server1), the initial connection is made to server2 instead of server1.

Failover with client affinities is seamless, if the following conditions are true:

- The connection is not in a transaction. That is, the failure occurs when the first SQL statement in the transaction is executed.
- There are no global temporary tables in use on the server.
- There are no open, held cursors.

When you use client affinities, you can specify that if the primary server returns to operation after an outage, connections return from an alternate server to the primary server on a transaction boundary. This activity is known as *failback*.

Configuring client affinities in non-Java clients for connection to DB2 for Linux, UNIX, and Windows

To enable support for client affinities in CLI and .NET applications, you set values in the db2dsdriver.cfg configuration file to indicate that you want to use client affinities, and to specify the primary and alternate servers.

The following table describes the settings in the db2dsdriver.cfg file for enabling client affinities for CLI and .NET applications.

Table 28. Settings to enable client affinities for CLI and .NET applications

Element in the acr section of the db2dsdriver configuration file	Values
enableAcr parameter	true

Table 28. Settings to enable client affinities for CLI and .NET applications (continued)

Element in the <code>acr</code> section of the <code>db2dsdriver</code> configuration file	Values
<code>maxAcrRetries</code> parameter	<p>The maximum number of connection attempts to each server in the list of alternate servers for automatic client reroute (ACR). The valid range is 0 - (the value of the <code>MAX_INT</code>).</p> <p>The value of the <code>DB2_MAX_CLIENT_CONNRETRIES</code> registry variable overrides the value of the <code>maxAcrRetries</code> parameter. If you do not set the <code>DB2_MAX_CLIENT_CONNRETRIES</code> registry variable or <code>maxAcrRetries</code> parameter, the <code>maxAcrRetries</code> parameter is set to 3. Setting the <code>maxAcrRetries</code> parameter to 0 disables ACR.</p>
<code>acrRetryInterval</code> parameter	<p>The number of seconds to wait between retries. The valid range is 0 to <code>MAX_INT</code>. The default is no wait (0).</p>
<code>affinityFailbackInterval</code> parameter	<p>The number of seconds to wait after the first transaction boundary to fail back to the primary server. Set this value if you want to fail back to the primary server. The default is 0, which means that no attempt is made to fail back to the primary server.</p>
<code>alternateserverlist</code>	<p><code><server></code> elements that identify the host name and port number for each server that is used for automatic client reroute through client affinities. One of the elements must identify the primary server. The presence of these elements does not activate automatic client reroute.</p>
<code>affinitylist</code>	<p><code><list></code> elements with <code>serverorder</code> attributes. The <code>serverorder</code> attribute value specifies a list of servers, in the order that they should be tried during automatic client reroute with client affinities. The servers in <code><list></code> elements must also be defined in <code><server></code> elements in the <code><alternateserverlist></code>. You can specify multiple <code><list></code> elements, each of which has different server orders. The presence of the <code><affinitylist></code> element does not activate automatic client reroute.</p>

Table 28. Settings to enable client affinities for CLI and .NET applications (continued)

Element in the acr section of the db2dsdriver configuration file	Values
client_affinity	<p>A <clientaffinitydefined> element or a <clientaffinityroundrobin> element that defines the order in which to try server connections for each client. When you include a <clientaffinitydefined> element, you define the server order by defining <client> elements, each of which specifies a <list> element that defines the server order. When you include a <clientaffinityroundrobin> element, you also specify <client> elements, but those <client> elements do not specify a <list> element. Instead, the order of the <client> elements, defines the server order. All clients that connect to a database must be specified within a <clientaffinitydefined> or a <clientaffinityroundrobin> element. In case of multiple network interface cards on a given client machine, client host name will be self discovered and matched with the configuration file entry, by CLI driver, to compute the affinity list. CLI driver will get all network interfaces and will try to match it with the host names available in the db2dsdriver configuration file. When a hostname without domain name is specified in db2dsdriver.cfg, CLI will try to resolve it using the default domain and will try to match with the discovered hostname. If the IP address is defined under client affinity section of the cfg file, the respective IP address will be discovered and matched (for hostname) with configuration file entry, by CLI driver, to compute the affinity list.</p>
clientaffinitydefined	<p><client> elements that define the server order for automatic client reroute for each client. Each <client> element contains a listname attribute that associates a client with a <list> element from the <affinitylist> element.</p>
clientaffinityroundrobin	<p><client> elements whose order in the <clientaffinityroundrobin> element defines the first server that is chosen for automatic client reroute. Each <client> element has an index. The first <client> element in the <clientaffinityroundrobin> element has index 0, the second <client> element has index 1, and so on. Suppose that the number of servers in the <alternateserverlist> element is n and the index in the <clientaffinityroundrobin> element of a <client> element is i. The first server to be tried is the server whose index in the <alternateserverlist> element is $i \bmod n$. The next server to be tried is the server whose index in the <alternateserverlist> element is $(i + 1) \bmod n$, and so on.</p>

The following restrictions apply to configuration of client affinities for CLI or .NET clients:

- If the total number of qualifying alternate servers for a given client is greater than 24, error SQL1042N occurs.
- Workload balancing cannot be enabled when client affinity is enabled. That is, if enableWLB is set to true, and the client_affinity element is specified, error SQL5162N occurs.
- If the required attributes are not specified in the <alternateserverlist>, <affinitylist> or <client_affinity> elements, error SQL5163N occurs.
- If client affinity is enabled, and the <alternateserverlist> element is empty, error SQL5164N occurs.
- If client affinity is enabled, and the host name for a client that is attempting to connect to a server is not in one of the <client_affinity> subgroups (<clientaffinitydefined> or <clientaffinityroundrobin>), or is in more than one of the subgroups, error SQL5164N occurs.
- For each client machine, there should be only one entry, either in <clientaffinitydefined> or <clientaffinityroundrobin> section. If there are entries in db2dsdriver.cfg where the same client machine has been specified by different host names, the error SQL5162N occurs.

Example of enabling client affinities for non-Java clients for DB2 for Linux, UNIX, and Windows connections

Before you can use client affinities for automatic client reroute in CLI or .NET applications, you need to include elements in the <acr> section of the db2dsdriver.cfg configuration file to indicate that you want to use client affinities, and to identify the primary and alternate servers.

The following example shows how to enable client affinities for failover without failback.

Suppose that your db2dsdriver configuration file looks like this:

```
<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <alternateserverlist>
      <server name="server1"
        hostname="v33ec067.svl.ibm.com"
        port="446">
      </server>
      <server name="server2"
        hostname="v33ec066.svl.ibm.com"
        port="446">
      </server>
      <server name="server3"
        hostname="v33ec065.svl.ibm.com"
        port="446">
      </server>
    </alternateserverlist>
    <affinitylist>
      <list name="list1" serverorder="server1,server2,server3">
      </list>
      <list name="list2" serverorder="server3,server2,server1">
      </list>
    </affinitylist>
  <clientaffinitydefined>
```



```

<!-- this section has specific defined affinities -->
  <client name="client1"
    hostname="appsrv1.svl.ibm.com"
    listname="list2">
  </client>
  <client name="client2"
    hostname="appsrv2.svl.ibm.com"
    listname="list1">
  </client>
</clientaffinitydefined>
<clientaffinityroundrobin>
  <client name="client3" hostname="appsrv3.svl.ibm.com">
    <!-- This entry is index 0. The number of servers is 3.
    0 mod 3 is 0, so the first that is tried
    during automatic client reroute is the server whose
    index in <alternateserverlist> is 0 (server1).
    The next server has index 1 mod 3, which is 1
    (server2). The final server has index 2 mod 3,
    which is 2 (server3). -->
  </client>
  <client name="client4" hostname="appsrv4.svl.ibm.com">
    <!-- This entry is index 1. The number of servers is 3.
    1 mod 3 is 1, so the first that is tried
    during automatic client reroute is the server whose
    index in <alternateserverlist> is 1 (server2).
    The next server has index 2 mod 3, which is 2
    (server3). The final server has index 3 mod 3,
    which is 0 (server1). -->
  </client>
</clientaffinityroundrobin>
</acr>
</database>

```

Suppose that a communication failure occurs during a connection from the client with host name `appsrv4.svl.ibm.com` (client4) to the server that is identified by `v33ec065.svl.ibm.com:446`. The following steps demonstrate the process that occurs for automatic client reroute with client affinities.

1. The driver tries to connect to `v33ec066.svl.ibm.com:446` (server2).
2. The connection to `v33ec066.svl.ibm.com:446` fails.
3. The driver waits two seconds.
4. The driver tries to connect to `v33ec065.svl.ibm.com:446` (server3).
5. The connection to `v33ec065.svl.ibm.com:446` fails.
6. The driver waits two seconds.
7. The driver tries to connect to `v33ec067.svl.ibm.com` (server1).
8. The connection to `v33ec067.svl.ibm.com` fails.
9. The driver waits two seconds.
10. The driver returns error code `SQL30081N`.

The following example shows how to enable client affinities for failover with failback.

Suppose that your `db2dsdriver` configuration file looks like this:

```

<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <parameter name="affinityFailbackInterval" value="300"/>
  <alternateserverlist>
    <server name="server1"

```



```

        hostname="v33ec067.svl.ibm.com"
        port="446">
</server>
<server name="server2"
        hostname="v33ec066.svl.ibm.com"
        port="446">
</server>
<server name="server3"
        hostname="v33ec065.svl.ibm.com"
        port="446">
</server>
</alternateserverlist>
<affinitylist>
  <list name="list1" serverorder="server1,server2,server3">
  </list>
  <list name="list2" serverorder="server3,server2,server1">
  </list>
</affinitylist>
<clientaffinitydefined>
<!-- this section has specific defined affinities -->
  <client name="client1"
        hostname="appsrv1.svl.ibm.com"
        listname="list2">
  </client>
  <client name="client2"
        hostname="appsrv2.svl.ibm.com"
        listname="list1">
  </client>
</clientaffinitydefined>
<clientaffinityroundrobin>
  <client name="client3" hostname="appsrv3.svl.ibm.com">
    <!-- This entry is index 0. The number of servers is 3.
         0 mod 3 is 0, so the first that is tried
         during automatic client reroute is the server whose
         index in <alternateserverlist> is 0 (server1).
         The next server has index 1 mod 3, which is 1
         (server2). The final server has index 2 mod 3,
         which is 2 (server3). -->
  </client>
  <client name="client4" hostname="appsrv4.svl.ibm.com">
    <!-- This entry is index 1. The number of servers is 3.
         1 mod 3 is 1, so the first that is tried
         during automatic client reroute is the server whose
         index in <alternateserverlist> is 1 (server2).
         The next server has index 2 mod 3, which is 2
         (server3). The final server has index 3 mod 3,
         which is 0 (server1). -->
  </client>
</clientaffinityroundrobin>
</acr>
</database>

```

Suppose that the database administrator takes the server that is identified by v33ec065.svl.ibm.com:446 down for maintenance after a connection is made from client appsrv2.svl.ibm.com (client2) to v33ec065.svl.ibm.com:446. The following steps demonstrate failover to an alternate server and failback to the primary server after maintenance is complete.

1. The driver successfully connects to v33ec065.svl.ibm.com:446 on behalf of client appsrv1.svl.ibm.com.
2. The database administrator brings down v33ec065.svl.ibm.com:446.
3. The application tries to do work on the connection.
4. The driver successfully fails over to v33ec066.svl.ibm.com:446.
5. After 200 seconds, the work is committed.

6. The driver tests whether the failback interval (300 seconds) has elapsed. It has not elapsed, so no failback occurs.
7. The application does more work on the connection to v33ec066.svl.ibm.com:446.
8. After 105 seconds, the work is committed.
9. The driver tests whether the failback interval (300 seconds) has elapsed. It has elapsed, so failback to v33ec065.svl.ibm.com:446 occurs.

Non-Java client support for high availability for connections to Informix servers

High-availability cluster support on IBM Informix servers provides high availability for client applications, through workload balancing and automatic client reroute. This support is available for applications that use Java clients (JDBC, SQLJ, or pureQuery), or non-Java clients (ODBC, CLI, .NET, OLE DB, PHP, Ruby, or embedded SQL).

For Java clients, you need to use IBM Data Server Driver for JDBC and SQLJ type 4 connectivity to take advantage of IBM Informix high-availability cluster support.

For non-Java clients, you need to use one of the following clients or client packages to take advantage of high-availability cluster support:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

Cluster support for high availability for connections to IBM Informix servers includes:

Automatic client reroute

This support enables a client to recover from a failure by attempting to reconnect to the database through any available server in a high-availability cluster. Reconnection to another server is called *failover*. You enable automatic client reroute on the client by enabling workload balancing on the client.

In an IBM Informix environment, primary and standby servers correspond to members of a high-availability cluster that is controlled by a Connection Manager. If multiple Connection Managers exist, the client can use them to determine primary and alternate server information. The client uses alternate Connection Managers only for the initial connection.

Failover for automatic client reroute can be *seamless* or *non-seamless*. With non-seamless failover, when the client application reconnects to an alternate server, the server always returns an error to the application, to indicate that failover (connection to the alternate server) occurred.

For Java, CLI, or .NET client applications, failover for automatic client reroute can be seamless or non-seamless. Seamless failover means that when the application successfully reconnects to an alternate server, the server does not return an error to the application.

Workload balancing

Workload balancing can improve availability of an IBM Informix high-availability cluster. When workload balancing is enabled, the client gets frequent status information about the members of a high-availability cluster. The client uses this information to determine the server to which the next

transaction should be routed. With workload balancing, IBM Informix Connection Managers ensure that work is distributed efficiently among servers and that work is transferred to another server if a server has a failure.

Connection concentrator

This support is available for Java applications that connect to IBM Informix. The connection concentrator reduces the resources that are required on IBM Informix database servers to support large numbers of workstation and web users. With the connection concentrator, only a few concurrent, active physical connections are needed to support many applications that concurrently access the database server. When you enable workload balancing on a Java client, you automatically enable the connection concentrator.

Client affinities

Client affinities is an automatic client reroute solution that is controlled completely by the client. It is intended for situations in which you need to connect to a particular primary server. If an outage occurs during the connection to the primary server, you use client affinities to enforce a specific order for failover to alternate servers.

Configuration of Informix high-availability support for non-Java clients

To configure a non-Java client application that connects to an Informix high-availability cluster, you must connect to an address that represents a Connection Manager. You must also set the properties that enable workload balancing and the maximum number of connections.

Before you can enable the IBM Data Server Driver for JDBC and SQLJ to connect to Informix database server for high availability, your installation must have one or more Connection Managers, a primary server, and one or more alternate servers.

The following table describes the basic settings, for non-Java applications.

Table 29. Basic settings to enable Informix high availability support in non-Java applications

Connection address	Value
database host ¹	The IP address of a Connection Manager. See “Setting server and port properties for connecting to a Connection Manager” on page 220.
database port ¹	The SQL port number of a Connection Manager. See “Setting server and port properties for connecting to a Connection Manager” on page 220.
database name ¹	The database name

Table 29. Basic settings to enable Informix high availability support in non-Java applications (continued)

Connection address	Value
Notes:	
<p>1. Depending on the DB2 product and driver you use, connection information can be defined in one of several possible sources.</p> <ul style="list-style-type: none"> • In scenario that involves CLI or open source application with IBM data server client, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, CLI driver uses that information to establish a connection. – Information from the database catalog. – If host and port information is not provided in the connection string of an application or database catalog, the driver searches for required information in the <code>db2cli.ini</code> file, and this information provided in the <code>db2cli.ini</code> file is used by CLI driver to establish a connection. – If host and port information is not provided in the connection string of an application, database catalog or the <code>db2cli.ini</code> file, CLI driver uses information in the <code>db2dsdriver.cfg</code> configuration file. • In scenario that involves CLI or open source application with IBM data server driver, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, CLI driver uses that information to establish a connection. – If host and port information is not provided in the connection string of an application, the driver searches for required information in the <code>db2cli.ini</code> file, and this information provided in the <code>db2cli.ini</code> file is used by CLI driver to establish a connection. – If host and port information is not provided in the connection string of the application or the <code>db2cli.ini</code> file, CLI driver uses information in the <code>db2dsdriver.cfg</code> configuration file. • In scenario that involves .NET application with IBM data server client, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, .NET data provider uses that information to establish a connection. – If host, port and database information provided through .NET object properties, .NET data provider uses that information to establish a connection. – Information from the database catalog. – If host and port information is not provided in the connection string of an application or the database catalog, .NET data provider uses information in the <code>db2dsdriver.cfg</code> configuration file. • In scenario that involves .NET application with IBM data server driver, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, .NET data provider uses that information to establish a connection. – If host, port and database information provided through .NET object properties, .NET data provider uses that information to establish a connection. – If host and port information is not provided in the connection string of an application or the database catalog, .NET data provider uses information in the <code>db2dsdriver.cfg</code> configuration file. 	

To fine-tune the workload balancing function of Informix high-availability support, additional properties are available. The additional properties for non-Java applications are listed in the following table.

Table 30. Properties for fine-tuning workload balancing support for connections from non-Java applications to Informix database server

Element in the db2dsdriver configuration file	Section in the db2dsdriver file	Description
enableWLB	<wlb>	Specifies whether workload balancing is enabled. Set this element to true to enable workload balancing.
maxTransportIdleTime	<wlb>	Specifies the maximum elapsed time before an idle transport is dropped. The default is 60. The minimum supported value is 0.
maxTransportWaitTime	<wlb>	Specifies the number of seconds that the client waits for a transport to become available. The default is 1. Specifying a value as -1 means unlimited wait time. The minimum supported value is 0.
maxTransports	<wlb>	Specifies the maximum number of connections that the requester can make to the high availability cluster. The default is -1 (unlimited). The minimum supported value is 1.
maxRefreshInterval	<wlb>	Specifies the maximum elapsed time in seconds before the server list is refreshed. The default is 10. The minimum supported value is 0.

If you must use workload balancing but your applications cannot handle the errors that are returned for automatic client reroute processing, set the following parameters in the `db2dsdriver.cfg` configuration file.

Table 31. Properties for enabling only Sysplex workload balancing for connections from non-Java applications to Informix database server

Element in the db2dsdriver configuration file	Section in the db2dsdriver file	Description	Value to set
enableWLB	<wlb>	Specifies whether workload balancing is enabled.	true. If the value of the enableAcr parameter is true, the connection manager retries the server connection. This parameter is supported by the DB2 for z/OS Version 9.0 and later server. The DB2 Version 9.7 Fix Pack 3 and later is required to use enableAcr parameter with connections to DB2 for z/OS Version 10.0 and later server. If the enableAcr and enableWLB parameter values are false, the server connection fails.
enableAcr	<acr>	Specifies whether automatic client reroute is enabled. For CLI or .NET applications, enabling automatic client reroute automatically enables seamless failover.	false.

Table 31. Properties for enabling only Sysplex workload balancing for connections from non-Java applications to Informix database server (continued)

Element in the db2dsdriver configuration file	Section in the db2dsdriver file	Description	Value to set
enableSeamlessAcr	<acr>	Specifies whether seamless failover is enabled. Seamless failover is supported for CLI, .NET, and embedded SQL applications. The default is true.	If the value of the enableAcr parameter is false, enableSeamlessAcr value is false, so you do not have to set it.

Setting server and port properties for connecting to a Connection Manager

To set the server and port number in the `db2dsdriver.cfg` configuration file for connecting to a Connection Manager, follow this process:

- If your high-availability cluster is using a single Connection Manager, set the server name and port number to the server name and port number of the Connection Manager.
- If your high-availability cluster is using more than one Connection Manager, follow this process:
 1. Specify the server name and port number of the main Connection Manager that you want to use.
 2. In the <acr> subsection in the database entry in the `db2dsdriver.cfg` configuration file, set the value of the **enableAlternateServerListFirstConnect** parameter to true.
 3. In the <alternateserverlist> parameter entries for the <acr> section of the `db2dsdriver.cfg` configuration file, set the server names and port numbers of alternative Connection Managers.

Example of enabling Informix high availability support in non-Java clients

Before you can use Informix high availability support in CLI, .NET, or embedded SQL applications that connect directly to Informix database server, you need to update the `db2dsdriver` configuration file with the appropriate settings, and connect to a Connection Manager.

The following example demonstrates setting up a CLI client to take advantage of Informix high availability support with one Connection Manager.

Before you can set up the client, you need to configure one or more high availability clusters that are controlled by Connection Managers.

Follow these steps to set up the client:

1. Create a `db2dsdriver.cfg` file with the basic settings for Informix high availability support. When you set `enableWLB` to true, you enable workload balancing and automatic client reroute capability.

```
<configuration>
  <dsncollection>
    <dsn alias="IDSCM1" name="IDSCM1" host="ids.cm1.ibm.com" port="446">
    </dsn>
  </dsncollection>
</databases>
```

```

<database name="IDSCM1" host="ids.cm1.ibm.com" port="446">
  <!-- database-specific parameters -->
  <wlb>
    <!-- Enable workload balancing to get
    automatic client reroute
    functionality -->
    <parameter name="enableWLB" value="true" />
    <!-- maxTransports represents the maximum number of transports -->
    <parameter name="maxTransports" value="80" />
  </wlb>
</database>
</databases>
<parameters>
  <parameter name="connectionLevelLoadBalancing" value="true"/>
</parameters>
</configuration>

```

- Suppose that the DSN definition for IDSCM1 provides connectivity information for a Connection Manager for database IDSCM1. In a CLI application, use code like this to connect to the Connection Manager:

```

...
SQLHDBC      hDbc      = SQL_NULL_HDBC;
SQLRETURN    rc        = SQL_SUCCESS;
SQLINTEGER   RETCODE = 0;
char         *ConnStrIn =
             "DSN=IDSCM1;PWD=mypass";
             /* dsn matches the database name in the configuration file */
char         ConnStrOut [200];
SQLSMALLINT  cbConnStrOut;
int          i;
char         *token;
...
/*****
/* Invoke SQLDriverConnect
*****/
RETCODE = SQLDriverConnect (hDbc
                           ,
                           NULL
                           ,
                           (SQLCHAR *)ConnStrIn
                           ,
                           strlen(ConnStrIn)
                           ,
                           (SQLCHAR *)ConnStrOut
                           ,
                           sizeof(ConnStrOut)
                           ,
                           &cbConnStrOut
                           ,
                           SQL_DRIVER_NOPROMPT);
...

```

Operation of the automatic client reroute feature when connecting to the Informix database server server from an application other than a Java application

Automatic client reroute support provides failover support when an IBM data server client loses connectivity to a server in an Informix high availability cluster. Automatic client reroute enables the client to recover from a failure by attempting to reconnect to the database through any available server in the cluster.

Automatic client reroute is enabled by default when workload balancing is enabled.

If automatic client reroute is enabled, the following process typically occurs when a client encounters a connection failure with an existing connection:

- The client attempts to execute an SQL statement using an existing connection and encounters a failure.

2. The client uses the server list that is returned by the Connection Manager to identify the server to access, and attempts to reconnect to the database.
3. If the automatic client reroute process can reconnect the application to the database, the client reconstructs the execution environment for the newly established connection. The error SQL30108N is returned to the application to indicate that the failed database connection has been recovered and that the transaction has been rolled back. The application is then responsible for further recovery, including repeating any work that was rolled back.

If the SQL statement that fails is the first SQL statement in the transaction, automatic client reroute with seamless failover is enabled, and the client is CLI or .NET, the driver replays the failed SQL operation as part of automatic client reroute processing. If the connection is successful, no error is reported to the application, and the transaction is not rolled back. The connectivity failure and subsequent recovery are hidden from the application.

4. If automatic client reroute is unable to reconnect to the database, the error SQL30081N is returned to the application. The application is then responsible for recovering from the connection failure (for example, by attempting to connect to the database by itself).

Automatic client reroute is also used when a client encounters a connection failure with a new connection. In this case, however, if reconnection is successful, no error is returned to the application to indicate that the failed database connection has been recovered. If reconnection fails, the error SQL30081N is returned.

If all the data, including the end of file (EOF) character, is returned in the first query block or in a subsequent fetch request, the CLI driver can perform seamless failover when you issue a COMMIT or ROLLBACK statement after the server becomes unreachable. For seamless failover to take place, the following conditions must be met:

- You must enable both the **enableAcr** and **enableSeamlessAcr** parameters.
- The cursor must have blocking enabled.
- The cursor must be either read only or forward only.

You can configure the **FetchBufferSize** keyword to ensure that the size of the result set that the CLI driver prefetches is sufficient to include the EOF character in the first query block. See the Related reference section for further details on **FetchBufferSize**.

In the non-seamless ACR environment, the **ReceiveTimeout** event triggers the ACR connection error (SQL30108N).

If the **QueryTimeout** parameter is set in the non-seamless ACR environment, following behaviors are observed:

- If the connection failure occurs before the query timeout event, the ACR connection error (SQL30108N with reason code 1, and failure code 1, 2 or 3) is returned to the application.
- If the **Interrupt** parameter is set to the default value of 1 and the query timeout event occurs, the SQL0952N error is returned to the application.
- If the **Interrupt** parameter is set to the value of 2 and the query timeout event occurs, the ACR connection error (SQL30108N with failure code 4 and error code 1) is returned to the application.

When the **Interrupt** parameter is set to the value of 2 and `SQLCancel()` API is explicitly sent from the application while SQL statement is being executed, the ACR connection error (SQL30108N failure code 4 and error code 2) is returned to the application.

If the **tcPIPConnectTimeout** parameter and the **memberConnectTimeout** parameter is set and the TCPIP Connection timeout or the member connection timeout event occurs in the non-seamless ACR environment, the CLI driver reconnects to a new available member but the ACR error (SQL30108N) is not returned to the application.

Operation of workload balancing for connections to Informix from non-Java clients

Workload balancing (also called transaction-level workload balancing) for connections to IBM Informix contributes to high availability by balancing work among servers in a high-availability cluster at the start of a transaction.

The following overview describes the steps that occur when a client connects to an IBM Informix Connection Manager, and workload balancing is enabled:

1. When the client first establishes a connection using the IP address of the Connection Manager, the Connection Manager returns the server list and the connection details (IP address, port, and weight) for the servers in the cluster. The server list is cached by the client. The default lifespan of the cached server list is 30 seconds.
2. At the start of a new transaction, the client reads the cached server list to identify a server that has untapped capacity, and looks in the transport pool for an idle transport that is tied to the under-utilized server. (An idle transport is a transport that has no associated connection object.)
 - If an idle transport is available, the client associates the connection object with the transport.
 - If, after a user-configurable timeout, no idle transport is available in the transport pool and no new transport can be allocated because the transport pool has reached its limit, an error is returned to the application.
3. When the transaction runs, it accesses the server that is tied to the transport.
4. When the transaction ends, the client verifies with the server that transport reuse is still allowed for the connection object.
5. If transport reuse is allowed, the server returns a list of SET statements for special registers that apply to the execution environment for the connection object. The client caches these statements, which it replays in order to reconstruct the execution environment when the connection object is associated with a new transport.
6. The connection object is then dissociated from the transport, if the client determines that it needs to do so.
7. The client copy of the server list is refreshed when a new connection is made, or every 30 seconds, or at the user-configured interval.
8. When workload balancing is required for a new transaction, the client uses the previously described process to associate the connection object with a transport.

Application programming requirements for high availability for connections from non-Java clients to Informix database server

Failover for automatic client reroute can be seamless or non-seamless. If failover for connections to Informix database server is not seamless, you need to add code to account for the errors that are returned when failover occurs.

If failover is non-seamless, and a connection is reestablished with the server, SQLCODE -4498 (for Java clients) or SQL30108N (for non-Java clients) is returned to the application. All work that occurred within the current transaction is rolled back. In the application, you need to:

- Check the reason code that is returned with the error. Determine whether special register settings on the failing data sharing member are carried over to the new (failover) data sharing member. Reset any special register values that are not current.
- Execute all SQL operations that occurred during the previous transaction.

The following conditions must be satisfied for seamless failover to occur during connections to IBM Informix databases:

- The application programming language is Java, CLI, or .NET.
- The connection is not in a transaction. That is, the failure occurs when the first SQL statement in the transaction is executed.
- The data server must allow transport reuse at the end of the previous transaction.
- All global session data is closed or dropped.
- There are no open held cursors.
- If the application uses CLI, the application cannot perform actions that require the driver to maintain a history of previously called APIs in order to replay the SQL statement. Examples of such actions are specifying data at execution time, performing compound SQL, or using array input.
- The application is not a stored procedure.
- Autocommit is not enabled. Seamless failover can occur when autocommit is enabled. However, the following situation can cause problems: Suppose that SQL work is successfully executed and committed at the data server, but the connection or server goes down before acknowledgment of the commit operation is sent back to the client. When the client re-establishes the connection, it replays the previously committed SQL statement. The result is that the SQL statement is executed twice. To avoid this situation, turn autocommit off when you enable seamless failover.

In addition, seamless automatic client reroute might not be successful if the application has autocommit enabled. With autocommit enabled, a statement might be executed and committed multiple times.

Client affinities for connections to Informix database server from non-Java clients

Client affinities is a client-only method for providing automatic client reroute capability.

Client affinities is available for applications that use CLI, .NET, or Java (IBM Data Server Driver for JDBC and SQLJ type 4 connectivity). All rerouting is controlled by the driver.

Client affinities is intended for situations in which you need to connect to a particular primary server. If an outage occurs during the connection to the primary server, you need to enforce a specific order for failover to alternate servers. You should use client affinities for automatic client reroute only if automatic client reroute that uses server failover capabilities does not work in your environment.

As part of configuration of client affinities, you specify a list of alternate servers, and the order in which connections to the alternate servers are tried. When client affinities is in use, connections are established based on the list of alternate servers instead of the host name and port number that are specified by the application. For example, if an application specifies that a connection is made to server1, but the configuration process specifies that servers should be tried in the order (server2, server3, server1), the initial connection is made to server2 instead of server1.

Failover with client affinities is seamless, if the following conditions are true:

- The connection is not in a transaction. That is, the failure occurs when the first SQL statement in the transaction is executed.
- There are no global temporary tables in use on the server.
- There are no open, held cursors.

When you use client affinities, you can specify that if the primary server returns to operation after an outage, connections return from an alternate server to the primary server on a transaction boundary. This activity is known as *failback*.

Configuring client affinities in non-Java clients for connection to Informix database server connections

To enable support for client affinities in CLI and .NET applications, you set values in the `db2dsdriver.cfg` configuration file to indicate that you want to use client affinities, and to specify the primary and alternate servers.

The following table describes the settings in the `db2dsdriver.cfg` file for enabling client affinities for CLI and .NET applications.

Table 32. Settings to enable client affinities for CLI and .NET applications

Element in the <code>acr</code> section of the <code>db2dsdriver</code> configuration file	Values
<code>enableAcr</code> parameter	true
<code>maxAcrRetries</code> parameter	<p>The maximum number of connection attempts to each server in the list of alternate servers for automatic client reroute (ACR). The valid range is 0 - (the value of the <code>MAX_INT</code>).</p> <p>The value of the <code>DB2_MAX_CLIENT_CONNRETRIES</code> registry variable overrides the value of the <code>maxAcrRetries</code> parameter. If you do not set the <code>DB2_MAX_CLIENT_CONNRETRIES</code> registry variable or <code>maxAcrRetries</code> parameter, the <code>maxAcrRetries</code> parameter is set to 3. Setting the <code>maxAcrRetries</code> parameter to 0 disables ACR.</p>

Table 32. Settings to enable client affinities for CLI and .NET applications (continued)

Element in the acr section of the db2dsdriver configuration file	Values
acrRetryInterval parameter	The number of seconds to wait between retries. The valid range is 0 to MAX_INT. The default is no wait (0).
affinityFailbackInterval parameter	The number of seconds to wait after the first transaction boundary to fail back to the primary server. Set this value if you want to fail back to the primary server. The default is 0, which means that no attempt is made to fail back to the primary server.
alternateserverlist	<server> elements that identify the host name and port number for each server that is used for automatic client reroute through client affinities. One of the elements must identify the primary server. The presence of these elements does not activate automatic client reroute.
affinitylist	<list> elements with serverorder attributes. The serverorder attribute value specifies a list of servers, in the order that they should be tried during automatic client reroute with client affinities. The servers in <list> elements must also be defined in <server> elements in the <alternateserverlist>. You can specify multiple <list> elements, each of which has different server orders. The presence of the <affinitylist> element does not activate automatic client reroute.

Table 32. Settings to enable client affinities for CLI and .NET applications (continued)

Element in the acr section of the db2dsdriver configuration file	Values
client_affinity	<p>A <clientaffinitydefined> element or a <clientaffinityroundrobin> element that defines the order in which to try server connections for each client. When you include a <clientaffinitydefined> element, you define the server order by defining <client> elements, each of which specifies a <list> element that defines the server order. When you include a <clientaffinityroundrobin> element, you also specify <client> elements, but those <client> elements do not specify a <list> element. Instead, the order of the <client> elements, defines the server order. All clients that connect to a database must be specified within a <clientaffinitydefined> or a <clientaffinityroundrobin> element. In case of multiple network interface cards on a given client machine, client host name will be self discovered and matched with the configuration file entry, by CLI driver, to compute the affinity list. CLI driver will get all network interfaces and will try to match it with the host names available in the db2dsdriver configuration file. When a hostname without domain name is specified in db2dsdriver.cfg, CLI will try to resolve it using the default domain and will try to match with the discovered hostname. If the IP address is defined under client affinity section of the cfg file, the respective IP address will be discovered and matched (for hostname) with configuration file entry, by CLI driver, to compute the affinity list.</p>
clientaffinitydefined	<p><client> elements that define the server order for automatic client reroute for each client. Each <client> element contains a listname attribute that associates a client with a <list> element from the <affinitylist> element.</p>
clientaffinityroundrobin	<p><client> elements whose order in the <clientaffinityroundrobin> element defines the first server that is chosen for automatic client reroute. Each <client> element has an index. The first <client> element in the <clientaffinityroundrobin> element has index 0, the second <client> element has index 1, and so on. Suppose that the number of servers in the <alternateserverlist> element is n and the index in the <clientaffinityroundrobin> element of a <client> element is i. The first server to be tried is the server whose index in the <alternateserverlist> element is $i \bmod n$. The next server to be tried is the server whose index in the <alternateserverlist> element is $(i + 1) \bmod n$, and so on.</p>

The following restrictions apply to configuration of client affinities for CLI or .NET clients:

- If the total number of qualifying alternate servers for a given client is greater than 128, error SQL1042N occurs.
- Workload balancing cannot be enabled when client affinity is enabled. That is, if enableWLB is set to true, and the client_affinity element is specified, error SQL5162N occurs.
- If the required attributes are not specified in the <alternateserverlist>, <affinitylist> or <client_affinity> elements, error SQL5163N occurs.
- If client affinity is enabled, and the <alternateserverlist> element is empty, error SQL5164N occurs.
- If client affinity is enabled, and the host name for a client that is attempting to connect to a server is not in one of the <client_affinity> subgroups (<clientaffinitydefined> or <clientaffinityroundrobin>), or is in more than one of the subgroups, error SQL5164N occurs.
- For each client machine, there should be only one entry, either in <clientaffinitydefined> or <clientaffinityroundrobin> section. If there are entries in db2dsdriver.cfg where the same client machine has been specified by different host names, the error SQL5162N occurs.

Example of enabling client affinities for non-Java clients for Informix database server connections

Before you can use client affinities for automatic client reroute in CLI or .NET applications, you need to include elements in the <acr> section of the db2dsdriver.cfg configuration file to indicate that you want to use client affinities, and to identify the primary and alternate servers.

The following example shows how to enable client affinities for failover without failback.

Suppose that your db2dsdriver configuration file looks like this:

```
<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <alternateserverlist>
      <server name="server1"
        hostname="v33ec067.svl.ibm.com"
        port="446">
      </server>
      <server name="server2"
        hostname="v33ec066.svl.ibm.com"
        port="446">
      </server>
      <server name="server3"
        hostname="v33ec065.svl.ibm.com"
        port="446">
      </server>
    </alternateserverlist>
    <affinitylist>
      <list name="list1" serverorder="server1,server2,server3">
      </list>
      <list name="list2" serverorder="server3,server2,server1">
      </list>
    </affinitylist>
  <clientaffinitydefined>
```

```

<!-- this section has specific defined affinities -->
  <client name="client1"
    hostname="appsrv1.svl.ibm.com"
    listname="list2">
  </client>
  <client name="client2"
    hostname="appsrv2.svl.ibm.com"
    listname="list1">
  </client>
</clientaffinitydefined>
<clientaffinityroundrobin>
  <client name="client3" hostname="appsrv3.svl.ibm.com">
    <!-- This entry is index 0. The number of servers is 3.
    0 mod 3 is 0, so the first that is tried
    during automatic client reroute is the server whose
    index in <alternateserverlist> is 0 (server1).
    The next server has index 1 mod 3, which is 1
    (server2). The final server has index 2 mod 3,
    which is 2 (server3). -->
  </client>
  <client name="client4" hostname="appsrv4.svl.ibm.com">
    <!-- This entry is index 1. The number of servers is 3.
    1 mod 3 is 1, so the first that is tried
    during automatic client reroute is the server whose
    index in <alternateserverlist> is 1 (server2).
    The next server has index 2 mod 3, which is 2
    (server3). The final server has index 3 mod 3,
    which is 0 (server1). -->
  </client>
</clientaffinityroundrobin>
</acr>
</database>

```

Suppose that a communication failure occurs during a connection from the client with host name `appsrv4.svl.ibm.com` (client4) to the server that is identified by `v33ec065.svl.ibm.com:446`. The following steps demonstrate the process that occurs for automatic client reroute with client affinities.

1. The driver tries to connect to `v33ec066.svl.ibm.com:446` (server2).
2. The connection to `v33ec066.svl.ibm.com:446` fails.
3. The driver waits two seconds.
4. The driver tries to connect to `v33ec065.svl.ibm.com:446` (server3).
5. The connection to `v33ec065.svl.ibm.com:446` fails.
6. The driver waits two seconds.
7. The driver tries to connect to `v33ec067.svl.ibm.com` (server1).
8. The connection to `v33ec067.svl.ibm.com` fails.
9. The driver waits two seconds.
10. The driver returns error code `SQL30081N`.

The following example shows how to enable client affinities for failover with failback.

Suppose that your `db2dsdriver` configuration file looks like this:

```

<database name="SAMPLE" host="v33ec065.svl.ibm.com" port="446">
  <acr>
    <parameter name="enableAcr" value="true"/>
    <parameter name="maxAcrRetries" value="1"/>
    <parameter name="acrRetryInterval" value="2"/>
    <parameter name="affinityFailbackInterval" value="300"/>
  <alternateserverlist>
    <server name="server1"

```

```

        hostname="v33ec067.svl.ibm.com"
        port="446">
</server>
<server name="server2"
        hostname="v33ec066.svl.ibm.com"
        port="446">
</server>
<server name="server3"
        hostname="v33ec065.svl.ibm.com"
        port="446">
</server>
</alternateserverlist>
<affinitylist>
  <list name="list1" serverorder="server1,server2,server3">
  </list>
  <list name="list2" serverorder="server3,server2,server1">
  </list>
</affinitylist>
<clientaffinitydefined>
<!-- this section has specific defined affinities -->
  <client name="client1"
        hostname="appsrv1.svl.ibm.com"
        listname="list2">
  </client>
  <client name="client2"
        hostname="appsrv2.svl.ibm.com"
        listname="list1">
  </client>
</clientaffinitydefined>
<clientaffinityroundrobin>
  <client name="client3" hostname="appsrv3.svl.ibm.com">
    <!-- This entry is index 0. The number of servers is 3.
         0 mod 3 is 0, so the first that is tried
         during automatic client reroute is the server whose
         index in <alternateserverlist> is 0 (server1).
         The next server has index 1 mod 3, which is 1
         (server2). The final server has index 2 mod 3,
         which is 2 (server3). -->
  </client>
  <client name="client4" hostname="appsrv4.svl.ibm.com">
    <!-- This entry is index 1. The number of servers is 3.
         1 mod 3 is 1, so the first that is tried
         during automatic client reroute is the server whose
         index in <alternateserverlist> is 1 (server2).
         The next server has index 2 mod 3, which is 2
         (server3). The final server has index 3 mod 3,
         which is 0 (server1). -->
  </client>
</clientaffinityroundrobin>
</acr>
</database>

```

Suppose that the database administrator takes the server that is identified by v33ec065.svl.ibm.com:446 down for maintenance after a connection is made from client appsrv2.svl.ibm.com (client2) to v33ec065.svl.ibm.com:446. The following steps demonstrate failover to an alternate server and failback to the primary server after maintenance is complete.

1. The driver successfully connects to v33ec065.svl.ibm.com:446 on behalf of client appsrv1.svl.ibm.com.
2. The database administrator brings down v33ec065.svl.ibm.com:446.
3. The application tries to do work on the connection.
4. The driver successfully fails over to v33ec066.svl.ibm.com:446.
5. After 200 seconds, the work is committed.

6. The driver tests whether the failback interval (300 seconds) has elapsed. It has not elapsed, so no failback occurs.
7. The application does more work on the connection to v33ec066.svl.ibm.com:446.
8. After 105 seconds, the work is committed.
9. The driver tests whether the failback interval (300 seconds) has elapsed. It has elapsed, so failback to v33ec065.svl.ibm.com:446 occurs.

Non-Java client support for high availability for connections to DB2 for z/OS servers

Sysplex workload balancing functionality on DB2 for z/OS servers provides high availability for client applications that connect directly to a data sharing group.

Sysplex workload balancing functionality provides workload balancing and automatic client reroute capability. This support is available for applications that use Java clients (JDBC, SQLJ, or pureQuery), or non-Java clients (ODBC, CLI, .NET, OLE DB, PHP, Ruby, or embedded SQL).

A Sysplex is a set of z/OS systems that communicate and cooperate with each other through certain multisystem hardware components and software services to process customer workloads. DB2 for z/OS subsystems on the z/OS systems in a Sysplex can be configured to form a data sharing group. With data sharing, applications that run on more than one DB2 for z/OS subsystem can read from and write to the same set of data concurrently. One or more coupling facilities provide high-speed caching and lock processing for the data sharing group. The Sysplex, together with the Workload Manager (WLM), dynamic virtual IP address (DVIPA), and the Sysplex distributor, allow a client to access a DB2 for z/OS subsystem over TCP/IP with network resilience, and distribute transactions for an application in a balanced manner across members within the data sharing group.

Central to these capabilities is a server list that the data sharing group returns on connection boundaries and optionally on transaction boundaries. This list contains the IP address and WLM weight for each data sharing group member. With this information, a client can distribute transactions in a balanced manner, or identify the member to use when there is a communication failure.

The server list is returned on the first successful connection to the DB2 for z/OS data server. After the client has received the server list, the client directly accesses a data sharing group member based on information in the server list.

DB2 for z/OS provides several methods for clients to access a data sharing group. The access method that is set up for communication with the data sharing group determines whether Sysplex workload balancing is possible. The following table lists the access methods and indicates whether Sysplex workload balancing is possible.

Table 33. Data sharing access methods and Sysplex workload balancing

Data sharing access method ¹	Description	Sysplex workload balancing possible?
Group access	<p>A requester uses the group's dynamic virtual IP address (DVIPA) to make an initial connection to the DB2 for z/OS location. A connection to the data sharing group that uses the group IP address and SQL port is always successful if at least one member is started. The server list that is returned by the data sharing group contains:</p> <ul style="list-style-type: none"> • A list of members that are currently active and can do work. • The WLM weight for each member. <p>The group IP address is configured by using the z/OS Sysplex distributor. To clients that are outside the Sysplex, the Sysplex distributor provides a single IP address that represents a DB2 location. In addition to providing fault tolerance, you can configure the Sysplex distributor to provide connection load balancing.</p>	Yes
Member-specific access	<p>A requester uses a location alias to make an initial connection to one of the members that is represented by the alias. A connection to the data sharing group that uses the group IP address and alias SQL port is always successful if at least one member is started. The server list that is returned by the data sharing group contains:</p> <ul style="list-style-type: none"> • A list of members that are currently active, can do work, and have been configured as an alias. • The WLM weight for each member. <p>The requester uses this information to connect to the member or members with the most capacity that are also associated with the location alias. Member-specific access is used when requesters are required to take advantage of Sysplex workload balancing among a subset of members of a data sharing group.</p>	Yes
Single-member access	<p>Single-member access is used when requesters are required to access only one member of a data sharing group. For single-member access, the connection uses the member-specific IP address.</p>	No
Note:		
<p>1. For more information about data sharing access methods, see http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z9.doc.dshare/src/tpc/db2z_tcpipaccessmethods.htm.</p>		

Sysplex workload balancing includes automatic client reroute: Automatic client reroute support enables a client to recover from a failure by attempting to reconnect to the database through any available member of a Sysplex. Reconnection to another member is called *failover*.

Starting in Version 10 fix pack 1 releases, alternate groups are an additional failover mechanism for automatic client rerouting when connectivity to the current group cannot be re-established. A *group* is a database created in a Sysplex data sharing environment. The database to which your application explicitly connects to is called the *primary group*.

For Java, CLI, or .NET client applications, failover for automatic client reroute can be *seamless* or *non-seamless*. Seamless failover means that when the application successfully reconnects to an alternate server or alternate group, the server does not return an error to the application.

Client direct connect support for high availability with a DB2 Connect server: Client direct connect support for high availability requires a DB2 Connect license, but does not require a DB2 Connect server. The client connects directly to DB2 for z/OS. If you use a DB2 Connect server, but set up your environment for client high availability, you cannot use some of the features that a direct connection to DB2 for z/OS provides. For example, you cannot use the transaction-level workload balancing or automatic client reroute capability that is provided by the sysplex.

Do not use client affinities: You should not use client affinities as a high availability solution for direct connections to DB2 for z/OS. Client affinities is not applicable to a DB2 for z/OS data sharing environment, because all members of a data sharing group can access data concurrently. A major disadvantage of client affinities in a data sharing environment is that if failover occurs because a data sharing group member fails, the member that fails might have retained locks that can severely affect transactions on the member to which failover occurs.

Configuration of Sysplex workload balancing and automatic client reroute for non-Java clients

To configure a client application other than a Java application that connects directly to a DB2 for z/OS server to use Sysplex workload balancing and automatic client reroute (ACR), set keyword values in the `db2dsdriver.cfg` file.

These keyword values specify a connection to an address that represents the data sharing group (for group access) or a subset of the data sharing group (for member-specific access) and that enables Sysplex workload balancing and automatic client reroute.

Always configure Sysplex workload balancing and automatic client reroute together. When you configure a client to use Sysplex workload balancing, automatic client reroute is also enabled by default. Therefore, you have to change keyword values that are related to automatic client reroute only to fine-tune automatic client reroute operation.

To configure the work load balancing (WLB) feature, you must set the **MaxTransports**, **ConnectionTimeout**, and **QueryTimeout** keyword values and leave remaining WLB and ACR keywords to their default value. When using the IBM Data Server Provide for .NET, the `CommandTimeout` property can be used instead of the **QueryTimeout** keyword.

For clients other than Java clients, use one of the listed clients or client packages to take advantage of Sysplex workload balancing:

- IBM Data Server Client
- IBM Data Server Runtime Client

- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

Important: To establish direct connections to a DB2 for z/OS data sharing group by using the Sysplex feature, you need either a DB2 Connect server product installation or DB2 Connect server license file in the license directory of the DB2 installation path.

Table 1 describes the basic configuration settings that are necessary to enable Sysplex workload balancing for applications other than Java applications.

Table 34. Basic settings to enable Sysplex workload balancing in applications other than Java applications

Data-sharing access method		
method	Client setting	Value
Group access	The enableWLB parameter in the <wlb> section of the db2dsdriver.cfg configuration file	true
	Connection address:	
	database host ¹	The group IP address or domain name of the data sharing group
	database port ¹	The SQL port number for the DB2 location
	database name ¹	The DB2 location name that is defined during installation
Member-specific access	The enableWLB parameter in the <wlb> section of the db2dsdriver.cfg configuration file	true
	Connection address:	
	database host ¹	The group IP address or domain name of the data sharing group
	database port ¹	The port number for the DB2 location alias
	database name ¹	The name of the DB2 location alias that represents a subset of the members of the data sharing group

Table 34. Basic settings to enable Sysplex workload balancing in applications other than Java applications (continued)

Data-sharing access		
method	Client setting	Value
Notes:		
<p>1. Depending on the DB2 product and driver you use, connection information can be defined in one of several possible sources.</p> <ul style="list-style-type: none"> • In scenario that involves CLI or open source application with IBM data server client, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, CLI driver uses that information to establish a connection. – Information from the database catalog. – If host and port information is not provided in the connection string of an application or database catalog, the driver searches for required information in the <code>db2cli.ini</code> file, and this information provided in the <code>db2cli.ini</code> file is used by CLI driver to establish a connection. – If host and port information is not provided in the connection string of an application, database catalog or the <code>db2cli.ini</code> file, CLI driver uses information in the <code>db2dsdriver.cfg</code> configuration file. • In scenario that involves CLI or open source application with IBM data server driver, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, CLI driver uses that information to establish a connection. – If host and port information is not provided in the connection string of an application, the driver searches for required information in the <code>db2cli.ini</code> file, and this information provided in the <code>db2cli.ini</code> file is used by CLI driver to establish a connection. – If host and port information is not provided in the connection string of the application or the <code>db2cli.ini</code> file, CLI driver uses information in the <code>db2dsdriver.cfg</code> configuration file. • In scenario that involves .NET application with IBM data server client, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, .NET data provider uses that information to establish a connection. – If host, port and database information provided through .NET object properties, .NET data provider uses that information to establish a connection. – Information from the database catalog. – If host and port information is not provided in the connection string of an application or the database catalog, .NET data provider uses information in the <code>db2dsdriver.cfg</code> configuration file. • In scenario that involves .NET application with IBM data server driver, connection information can be obtained from following sources: <ul style="list-style-type: none"> – If host, port, and database information is provided in a connection string of an application, .NET data provider uses that information to establish a connection. – If host, port and database information provided through .NET object properties, .NET data provider uses that information to establish a connection. – If host and port information is not provided in the connection string of an application or the database catalog, .NET data provider uses information in the <code>db2dsdriver.cfg</code> configuration file. 		

To fine-tune the Sysplex workload balancing, additional properties are available. The additional properties for applications other than Java applications are listed in Table 2.

Table 35. Properties for fine-tuning Sysplex workload balancing for direct connections from applications other than Java applications to DB2 for z/OS

Element in the db2dsdriver.cfg configuration file	Section in the db2dsdriver.cfg file	Description
maxTransportIdleTime	<wlb>	Specifies the maximum elapsed time in seconds before an idle transport is dropped. The default is 60. The minimum supported value is 0.
maxTransportWaitTime	<wlb>	Specifies the number of seconds that the client waits for a transport to become available. The default value is 1 second. Specifying a value as -1 means unlimited wait time. The minimum supported wait time is 0.
maxTransports	<wlb>	Specifies the maximum number of physical connections (or transports) that the requester can make to the data sharing group. A maximum of one transport per member can be opened for each application connection. For example, 25 application connections accessing a 6 member group can open a maximum of 150 transports to the group. That means 6 transports, one to each member, can be opened for each application connection. The default is 1000. The value of -1 specifies the use of as many transports as necessary based on the number of application connections and members in use.
maxRefreshInterval	<wlb>	Specifies the maximum elapsed time in seconds before the server list is refreshed. The default is 10. The minimum supported value is 0.

Automatic client reroute capability is enabled in a client by default when the Sysplex workload balancing (WLB) feature is enabled. At the first successful connection to the server, the client obtains a list of all available servers (server list) from the connected group. The server list contains all available servers with capacity to run work. The server list may not contain all members of the DB2 data sharing group as only the available servers with capacity to run work are included in the list. Other than the case where server list is used to connect to the alternate servers, connection to a DB2 data sharing group uses the z/OS Sysplex distributor configured with the distributed dynamic virtual IP address. The z/OS Sysplex distributor uses the dynamic virtual IP address (VIPA) and automatic VIPA takeover to distribute incoming connection among the DB2 subsystems within the Sysplex environment. The z/OS Sysplex feature ensures the high availability of a DB2 data sharing group.

You can set configuration keywords and registry variables in the db2dsdriver.cfg file to control automatic client reroute behavior. The keyword descriptions in Table 3 are for the case in which client affinities are not enabled. If the db2dsdriver.cfg file changes, your CLI application can invoke the SQLReloadConfig function to validate the entries for all alternate servers within the <acr> section.

Table 36. Settings to control automatic client reroute behavior

Element in the <acr> section of the db2dsdriver configuration file	Value
enableAcr parameter	Specifies whether automatic client reroute is in effect. The default is true.

Table 36. Settings to control automatic client reroute behavior (continued)

Element in the <acr> section of the db2dsdriver configuration file	Value
enableSeamlessAcr parameter	Specifies whether seamless failover can occur. If the enableAcr parameter is set to true, the default for the enableSeamlessAcr parameter is true. The enableSeamlessACR parameter applies only to the members within a group or cluster. When you enable automatic client reroute to a DB2 for z/OS data sharing group, you must ensure that the enableSeamlessACR parameter is set to the default value of true, and the application can handle the SQL30108N exception.
acrRetryInterval parameter	The number of seconds to wait between consecutive connection attempts. The valid range is 0 - maximum integer value. The value of the DB2_CONNRETRIES_INTERVAL registry variable overrides the value of the acrRetryInterval parameter. The default is 0 (no wait) if you do not set the DB2_CONNRETRIES_INTERVAL registry variable. If you enable automatic client reroute to a DB2 for z/OS data sharing group, the default value of no wait is recommended.
maxAcrRetries parameter	Specifies the maximum number of connection attempts for automatic client reroute. The valid range is 0 to MAX_INT. The value of the DB2_MAX_CLIENT_CONNRETRIES registry variable overrides this value. If you do not set the DB2_MAX_CLIENT_CONNRETRIES registry variable or maxAcrRetries parameter, by default, one retry through the server list and the group IP address is attempted. Setting this value to 0 disables ACR.

The registry variables in Table 4 control retry behavior for automatic client reroute.

Table 37. Registry variables to control automatic client reroute retry behavior

Registry variable	Value
DB2_MAX_CLIENT_CONNRETRIES	The maximum number of connection retries for automatic client reroute. The default is 30 if you set the DB2_CONNRETRIES_INTERVAL variable.
DB2_CONNRETRIES_INTERVAL	The number of seconds between consecutive connection retries. The default is 10 if you set the DB2_MAX_CLIENT_CONNRETRIES variable.

When enabling automatic client reroute to a DB2 for z/OS data sharing group, set the **maxAcrRetries** parameter. If you do not set both the **DB2_MAX_CLIENT_CONNRETRIES** and **DB2_CONNRETRIES_INTERVAL** registry variables or do not set both the **maxAcrRetries** and **acrRetryInterval** parameters, automatic client reroute attempts to connect to a z/OS group for up to 10 minutes, with no wait between attempts.

For CLI, OLE DB, ADO.NET, and embedded SQL applications, there are three connection timeout keywords, which can be set in the `db2dsdriver.cfg` file:

tcpipConnectionTimeout

The **tcpipConnectionTimeout** keyword specifies the number of seconds before an attempt to open a socket fails. Do not use this keyword with automatic client reroute.

ConnectionTimeout

The **ConnectionTimeout** keyword specifies the number of seconds that a client application waits for a connection to a DB2 for z/OS data sharing group to be established.

MemberConnectTimeout

The **MemberConnectTimeout** keyword specifies the number of seconds that a client application waits before being routed to the next IP address in the server list. When you enable automatic client reroute for connections to a DB2 for z/OS data sharing group, you should use the **MemberConnectTimeout** keyword to manage the time to wait before rerouting. The default value of the **MemberConnectTimeout** keyword is 1 second. In most cases, the default value is adequate, but you may require higher **MemberConnectTimeout** value to prevent frequent network time-outs.

The **ConnectionTimeout** keyword setting takes precedence over the **tcpipConnectTimeout** and **MemberConnectTimeout** keyword settings.

If you must use Sysplex workload balancing but your applications cannot handle the errors that are returned for automatic client reroute processing, set the following parameters in the `db2dsdriver.cfg` configuration file.

Table 38. Parameters for enabling only Sysplex workload balancing for connections from applications other than Java applications to DB2 for z/OS

Element in the <code>db2dsdriver.cfg</code> configuration file	Section in the <code>db2dsdriver.cfg</code> file	Description	Value to set
connectionLevelLoadBalancing	<database>	Specifies whether connection-level load balancing is in effect. By default, if the enableWLB configuration parameter is set to true , the connectionLevelLoadBalancing parameter is set to true as well. Otherwise, the default value of the connectionLevelLoadBalancing parameter is false .	true
enableWLB	<wlb>	Specifies whether workload balancing is enabled. By default, the enableWLB parameter is set to false .	true
enableAcr	<acr>	Specifies whether automatic client reroute is enabled. For CLI or .NET applications, enabling automatic client reroute automatically enables seamless failover. By default, if the enableWLB parameter is set to true , the enableAcr parameter is set to true as well. Otherwise, the default is false . If application cannot handle the seamless failover exception (SQL30108N), set the enableAcr parameter to false , if the enableWLB parameter is set to true .	true
enableSeamlessAcr	<acr>	Specifies whether seamless failover is enabled. Seamless failover is supported only for Java, CLI, and .NET applications. By default, the enableSeamlessAcr parameter is set to same value as the enableAcr configuration parameter.	true
enableAlternateGroupSeamlessACR		Specifies seamless or non-seamless failover behavior across groups. The default is false . To set this parameter to true , you must also set the enableSeamlessACR configuration parameter to true . Setting the enableAlternateGroupSeamlessACR parameter to true does not affect the setting of the enableSeamlessACR parameter. If a connection is established to a server in the alternategroup parameter, the rules for seamless or non-seamless behavior still apply.	true

Example of enabling DB2 for z/OS Sysplex workload balancing and automatic client reroute in non-Java client applications

Before you can use Sysplex workload balancing and automatic client reroute in applications other than Java applications that connect directly to DB2 for z/OS servers, you need to update the `db2dsdriver.cfg` configuration file with the appropriate settings, and connect to a data sharing group.

Before you can set up the client, you need to configure the listed server software:

- WLM for z/OS

For workload balancing to work efficiently, DB2 work needs to be classified. Classification applies to the first non-SET SQL statement in each transaction.

Among the areas by which you need to classify the work are:

- Authorization ID
- Client info properties
- Stored procedure name

The stored procedure name is used for classification only if the first statement that is issued by the client in the transaction is an SQL CALL statement.

For a complete list of classification attributes, see the information about classification attributes at the succeeding URL: http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2z10.doc.perf/src/tpc/db2z_classificationattributes.htm

- DB2 for z/OS, set up for data sharing

Automatic client reroute capability is enabled in a client by default when the Sysplex workload balancing (WLB) feature is enabled. At the first successful connection to the server, the client obtains a list of all available servers (server list) from the connected group. The server list contains all available servers with capacity to run work. The server list may not contain all members of the DB2 data sharing group as only the available servers with capacity to run work are included in the list. Other than the case where server list is used to connect to the alternate servers, connection to a DB2 data sharing group uses the z/OS Sysplex distributor configured with the distributed dynamic virtual IP address. The z/OS Sysplex distributor uses the dynamic virtual IP address (VIPA) and automatic VIPA takeover to distribute incoming connection among the DB2 subsystems within the Sysplex environment. The z/OS Sysplex feature ensures the high availability of a DB2 data sharing group.

You can fine-tune default automatic client reroute feature by modifying these items:

Automatic client reroute characteristic	db2dsdriver.cfg configuration keyword	Desired value
Number of times to try connecting to the alternate server	<code>maxAcrRetries</code>	< 6
Number of seconds to wait between tries	<code>acrRetryInterval</code>	0 (default)

The example demonstrates how to set up a client application other than a Java application to take advantage of Sysplex and automatic client reroute high availability support.

1. Create a db2dsdriver.cfg file with the basic settings for Sysplex support and automatic client reroute. When you set enableWLB and enableAcr to true, you enable Sysplex workload balancing and automatic client reroute capabilities.

```

<configuration>
  <dsncollection>
    <dsn alias="DSGROUP1" name="DSGROUP1"
      host="db2a.sysplex1.ibm.com" port="446">
    </dsn>
  </dsncollection>
  <database name="DSGROUP1" host="db2a.sysplex1.ibm.com" port="446">
    <!-- database-specific parameters -->
    <wlb>
      <!-- Enable Sysplex workload balancing to get
      automatic client reroute
      functionality -->
      <parameter name="enableWLB" value="true" />
      <!-- maxTransports represents the maximum number of transports -->
      <parameter name="maxTransports" value="80" />
    </wlb>
    <acr>
      <parameter name="enableAcr" value="true">
      </parameter>
      <parameter name="maxAcrRetries" value="5">
      </parameter>
      <parameter name="acrRetryInterval" value="0">
      </parameter>
    </acr>
  </database>
</configuration>

```

2. Suppose that database name DSGROUP1 represents a data sharing group that is set up for group access. In a CLI application, use code like the succeeding code to connect to the data sharing group:

```

...
SQLHDBC      hDbc      = SQL_NULL_HDBC;
SQLRETURN    rc        = SQL_SUCCESS;
SQLINTEGER   RETCODE = 0;
char         *ConnStrIn =
              "DSN=DSGROUP1;PWD=mypass";
              /* dsn matches the database name in the configuration file */
char         ConnStrOut [200];
SQLSMALLINT  cbConnStrOut;
int          i;
char         *token;
...
/*****
/* Invoke SQLDriverConnect */
*****/
RETCODE = SQLDriverConnect (hDbc
                          ,
                          NULL
                          ,
                          (SQLCHAR *)ConnStrIn
                          ,
                          strlen(ConnStrIn)
                          ,
                          (SQLCHAR *)ConnStrOut
                          ,
                          sizeof(ConnStrOut)
                          ,
                          &cbConnStrOut
                          ,
                          SQL_DRIVER_NOPROMPT);
...

```

Operation of Sysplex workload balancing for connections from non-Java clients to DB2 for z/OS servers

Sysplex workload balancing (also called transaction-level workload balancing) for connections to DB2 for z/OS contributes to high availability by balancing work among members of a data sharing group at the start of a transaction.

The following overview describes the steps that occur when a client connects to a DB2 for z/OS Sysplex, and Sysplex workload balancing is enabled:

1. When the client first establishes a connection using the sysplex-wide IP address called the group IP address, or when a connection is reused by another connection object, the server returns member workload distribution information.

The default lifespan of the cached server list is 30 seconds.

2. At the start of a new transaction, the client reads the cached server list to identify a member that has untapped capacity, and looks in the transport pool for an idle transport that is tied to the under-utilized member. (An idle transport is a transport that has no associated connection object.)
 - If an idle transport is available, the client associates the connection object with the transport.
 - If, after a user-configurable timeout, no idle transport is available in the transport pool and no new transport can be allocated because the transport pool has reached its limit, an error is returned to the application.
3. When the transaction runs, it accesses the member that is tied to the transport.
4. When the transaction ends, the client verifies with the server that transport reuse is still allowed for the connection object.
5. If transport reuse is allowed, the server returns a list of SET statements for special registers that apply to the execution environment for the connection object.

The client caches these statements, which it replays in order to reconstruct the execution environment when the connection object is associated with a new transport.

6. The connection object is then disassociated from the transport.
7. The client copy of the server list is refreshed when a new connection is made, or every 30 seconds.
8. When workload balancing is required for a new transaction, the client uses the same process to associate the connection object with a transport.

Operation of the automatic client reroute feature for an application other than a Java application to the DB2 for z/OS server

The automatic client reroute (ACR) feature provides the failover support when an application loses connectivity to a member of the DB2 for z/OS data sharing group.

The ACR feature enables an application to recover from a connection failure by reconnecting to the database through an available member of the DB2 for z/OS data sharing group that has the Sysplex feature enabled.

The ACR feature is enabled by default when the Sysplex workload balancing feature is enabled.

Client support for the ACR feature is available in the IBM data server clients that have a DB2 Connect license. The DB2 Connect server product is not required for the ACR feature.

The following example demonstrates the ACR operation when an application other than a Java application connects to the DB2 for z/OS data sharing group with the Sysplex workload balancing feature enabled:

1. As part of the response to a COMMIT request from the client, the data server returns:
 - An indicator that specifies whether transports can be reused. Transports can be reused if there are no session resources remaining, such as held cursors.
 - The SET statements that the client can use to replay the connection state during transport reuse. The SET statements are also known as special registers.
2. If the first SQL statement in a transaction fails:
 - No error is reported to the application.
 - The failing SQL statement is executed again. The failing SQL statement is executed on the next available member based on the returned server list. The number of times that the statement is retried and the duration between retry attempts are based on the **maxAcrRetries** and **acrRetryInterval** parameters.
 - The SET statements that are associated with the connection are replayed to restore the connection state.
3. If an SQL statement that is not the first SQL statement in a transaction fails, and transports can be reused:
 - The transaction is rolled back.
 - The application is reconnected to the data server.
 - The SET statements (special registers) that are associated with the connection are replayed to restore the connection state.
 - The SQL30108N error is returned to the application to notify it of the rollback and successful reconnection. The application needs to include a code to retry the failed transaction.
4. If an SQL statement that is not the first SQL statement in a transaction fails, and transports cannot be reused:
 - The special register settings are returned to the last COMMIT point.
 - The SQL30081N error is returned to the application to notify it that reconnection was unsuccessful. The application needs to reconnect to the data server, reestablish the connection state, and retry the failed transaction.
5. If all the data, including the end of file (EOF) character, is returned in the first query block or in a subsequent fetch request, the CLI driver can perform seamless failover when you issue a COMMIT or ROLLBACK statement after the server becomes unreachable. If you declare any session resources in a translation, seamless ACR connections are changed into non-seamless ACR connections. The session resources include:
 - Open and held cursor
 - Open cursors with locators
 - Declared global temporary table
 - Accelerated Query

You can configure the **FetchBufferSize** keyword to ensure that the size of the result set that the CLI driver prefetches is sufficient to include the EOF character in the first query block. See the Related reference section for further details on the **FetchBufferSize** parameter.

6. If connections to all members of the data sharing member list have been tried, and none have succeeded, a connection is tried using the DB2 group-wide IP address that is associated with the data sharing group, to determine whether any members are now available.

In V10.1 Fix Pack 2 and later fix packs, when connecting to the DB2 for z/OS data sharing group with the workload balance (WLB) feature enabled, the behavior of the non-seamless ACR feature has changed:

- The CLI driver does not associate a transport with the connection until the application issues an SQL statement or a SET statement. When the application issues an SQL statement or a SET statement, the CLI driver allocates a transport and sets special registers to the COMMIT point of the last transaction.
- The SQL30108N error is returned to the application if the CLI driver fails to reconnect to members of the primary group and must switch to the alternate group. The error is returned a second time when the alternate group is specified in the `db2dsdriver.cfg` file with the **alternategroup** parameter and the **enableAlternateGroupSeamlessAcr** is set to FALSE. The first SQL30108N error with reason code 2 is returned when the existing connection to a member in the current group fails. The second SQL30108N error with reason code 4 is returned when all the connection attempts to all members in the existing primary group fail. The application can then resubmit the SET statement or the SQL statement again for the second time if reconnection to the alternate group is warranted. The CLI driver tracks the failed member on the same connection handle when the ACR connection error (SQL30108N) is returned to avoid resubmitting the statement to the failed member.

Note: The SQL30108N error is not returned twice in the following scenarios:

- When the DB2 Connect server is used as a gateway.
- When the ACR feature is explicitly enabled without enabling the WLB feature.

When connecting to the DB2 for z/OS data sharing group, you should not disable the seamless ACR feature and the WLB feature unless directed by IBM support.

In V10.1 Fix Pack 2 and later fix packs, the following example demonstrates the non-seamless ACR operation when the Sysplex workload balancing feature is enabled:

1. As part of the response to a COMMIT request from the client, the data server returns the SET statements (special registers) that the client replays when the transport is associated at the next SQL statement submission or the SET statement submission based on the WLB routing decisions.
2. If the SQL statement in a transaction fails, the ACR connection error (SQL30108N with reason code 2) is returned to the application but the CLI driver does not look for a new transport. Any session resources that were created in the failed connection are dropped. The session resources include:
 - Open and held cursor
 - Open cursors with locators
 - Declared global temporary table
 - Accelerated Query
3. If the application submits the SET statement or the SQL statement, the CLI driver attempts to obtain a new transport to connect to the next available member in the same group.
4. If the **enableAlternateGroupSeamlessACR** parameter is set to FALSE and the CLI driver cannot reconnect to any of the members in the primary group, the second ACR connection error (SQL30108N with reason code 4) is returned to the application.

5. If the application submits the SET statement or the SQL statement again for the second time, the CLI driver attempts to obtain a new transport to connect to the next available member in the alternate group.
6. When the CLI driver successfully reconnects to a new member, the SET statements (special registers) that were returned from the last COMMIT point are replayed, followed by the SET statement or SQL statement submitted by the application. Any session resources that existed in a previous failed connection that are required must be re-created. If the reconnection is not successful, the communication error (SQL30081N) is returned to the application.

In the non-seamless ACR environment, the **ReceiveTimeout** event triggers the ACR connection error (SQL30108N).

If the **QueryTimeout** parameter is set in the non-seamless ACR environment, the following behaviors are observed:

- If the connection failure occurs before the query timeout event, the ACR connection error (SQL30108N with reason code 2 or 4, and failure code 1, 2 or 3) is returned to the application.
- If the **Interrupt** parameter is set to the value of 2 and the query timeout event occurs, the ACR connection error (SQL30108N with failure code 4 and error code 1) is returned to the application.

When the **Interrupt** parameter is set to the value of 2 and the SQLCancel() API is explicitly sent from the application while the SQL statement is being executed, the ACR connection error (SQL30108N failure code 4 and error code 2) is returned to the application.

Operation of transaction-level workload balancing for connections to the DB2 for z/OS data sharing group

Transaction-level workload balancing for connections to the DB2 for z/OS database contributes to high availability by balancing work among servers in a DB2 for z/OS data sharing group at the start of a transaction.

When a client connects to a DB2 for z/OS server and transaction-level workload balancing is enabled, the succeeding steps occur:

1. When the client first establishes a connection to the DB2 for z/OS data sharing group that is using the distributed group IP address, the client returns a server list with the connection details (IP address, port, and weight) for the members of the DB2 for z/OS data sharing group.

The server list is cached by the client. The default life span of the cached server list is 30 seconds.

2. At the start of a new transaction, the client reads the cached server list to identify a server that has unused capacity, and looks in the transport pool for an idle transport that is tied to the under-utilized server. An idle transport is a transport that has no associated connection object.
 - If an idle transport is available, the client associates the connection object with the transport.
 - If, after a user-configurable timeout period (**db2.jcc.maxTransportObjectWaitTime** for a Java client or **maxTransportWaitTime** for other clients), no idle transport is available in the transport pool and no new transport can be allocated because the transport pool reached its limit, an error is returned to the application.
3. When the transaction runs, it accesses the server that is tied to the transport.

4. When the transaction ends, the client verifies with the server that transport reuse is still allowed for the connection object.
5. If transport reuse is allowed, the server returns a list of SET statements for special registers that apply to the execution environment for the connection object.
The client caches these statements, which it replays in order to reconstruct the execution environment when the connection object is associated with a new transport.
6. The connection object is then dissociated from the transport, if the client determines that it needs to do so.
7. The client copy of the server list is refreshed when a new connection is made, every 30 seconds, or each user-configured interval.
8. When transaction-level workload balancing is required for a new transaction, the client uses the previously described process to associate the connection object with a transport.

Alternate groups for connections to DB2 for z/OS servers from non-Java clients

To improve high availability for non-Java clients in Version 9.7 Fix Pack 5 or later fix pack releases, use alternate groups as an additional failover mechanism for automatic client rerouting when connectivity to the current group cannot be re-established.

By default, non-Java clients have the automatic client reroute (ACR) enabled. This capability provides automatic failover to alternate servers within the current group when connectivity to a server cannot be re-established.

In addition to this ACR capability, you can define *alternate groups* as failover targets when connectivity to the current group cannot be established. To define alternate groups for non-Java clients:

- Define one <database> element inside the <alternategroup> element in the <acr> section of the db2dsdriver.cfg file. Do not specify <parameter> elements inside the <database> element, parameter settings are inherited from the primary group.
- If you want to suppress error messages from failover connections to the alternate group, set the enableAlternateGroupSeamlessACR parameter to true in <alternategroup> element.

For DB2 for z/OS, you can define only one database in the alternate group. If you define more than one DB2 for z/OS, the connection is terminated and the client returns an error.

When a non-Java client is connected to an alternate group, all the connection settings and the parameter settings for the <database> element in the primary group are inherited by the connection to the database in the alternate group.

After a non-Java client is connected to a database in the alternate group, no failback to the primary group is provided. To connect to the primary group again, the application or client must be restarted.

Alternate groups are only supported for ACR and workload balancing. If client affinities is configured, alternate group definitions are ignored.

Examples

Here is an example of alternate group definitions in the `db2dsdriver.cfg` file:

```
<dsncollection>
  <dsn alias="mydsn2" name="mydb2" host="myserver2.ibm.com" port="5912">
    ...
</dsncollection>

<databases>
  <database name="mydb2" host="myserver2.ibm.com" port="5912">
    <parameter name="IsolationLevel" value="4"/>
    ...
    <wlb>
      <parameter name="enableWLB" value="true"/>
    </wlb>
    <acr>
    ...
    <alternategroup>
      <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
      <database name="mydb3" host="myserver3.ibm.com" port="5912">
        </database>
      </alternategroup>
    </acr>
  </database>

  <database name="mydb3" host="myserver3.ibm.com" port="5912">
    <parameter name="IsolationLevel" value="2"/>
    ...
    <acr>
      <parameter name="enableACR" value="true"/>
    ...
    <alternategroup>
      <parameter name="enableAlternateGroupSeamlessACR" value="true"/>
      <database name="mydb4" host="myserver4.ibm.com" port="5912">
        </database>
      </alternategroup>
    </acr>
    ...
  </database>
</databases>
```

The following example scenarios demonstrate how automatic client rerouting works for alternate groups. The details about ACR failover to the current group are not covered in these scenarios to focus on the alternate groups failover details. These scenarios use the `db2dsdriver.cfg` sample that is described in the previous paragraph.

First connection to the primary

In this example, after a non-Java client fails to connect to the primary group on its first attempt, the connection to an alternate group is attempted:

1. The client fails to connect to the *mydb2* database.
2. The client tries to connect to the alternate group listed in the `<alternategroup>` section of the `db2dsdriver.cfg` file in the order specified in this file.
 - a. The client successfully connects to the *mydb3* database.

Subsequent connection after the loss of existing connection to the primary server

In this example, after a non-Java client loses its connection to the *mydb3* database, the following connection attempts are made:

1. The client fails to connect to all the members returned in the server list.
2. The client fails to connect to the *mydb3* database once again.

3. The client tries to connect to an alternate group listed in the <alternategroup> section of the `db2dsdriver.cfg` file.
 - a. The client successfully connects to the *mydb4* database.

When connecting to DB2 for z/OS server, only one <database> element is allowed under the <alternategroup> section.

Existing connection to an alternate group

A non-Java client fails to connect to the *mydb2* database, automatic client reroute failover to alternate servers in the current group also fails, and then it successfully connects to the *mydb3* database in the alternate group.

If communication error is encountered, you must restart the client or the application must try connecting to the primary group again.

Application programming requirements for the automatic client reroute environment when connecting to the DB2 for z/OS servers

Failover scenario for the automatic client reroute feature can be seamless or non-seamless. If failover for connections to the DB2 for z/OS data sharing group is not seamless, you must add a code to account for the errors that are returned when failover occurs.

If failover is not seamless, and a connection is reestablished with the server, the ACR connection error (SQL30108N) is returned to the application. All work that occurred within the current transaction is rolled back. In the application, you must:

- Check the reason code that is returned with the -30108 error to determine whether special register settings on the failing data sharing member are carried over to the new (failover) data sharing member. Reset any special register values that are not current.
- Re-create any session resources that were created in the previous failed connection. The session resources can include:
 - An open and held cursor
 - Open cursors with locators
 - A declared global temporary table
 - An accelerated query
- Execute all SQL operations that occurred since the previous commit operation.

The following conditions must be satisfied for seamless failover to occur for direct connections to DB2 for z/OS:

- The application language is Java, CLI, or .NET.
- The connection is not in a transaction. That is, the failure occurs when the first SQL statement in the transaction is executed.
- The data server allows transport reuse at the end of the previous transaction. An exception to this condition is if transport reuse is not granted because the application was bound with `KEEPDYNAMIC(YES)`.
- All global session data is closed or dropped.
- There are no open, held cursors.
- If the application uses CLI, the application cannot perform actions that require the driver to maintain a history of previously called APIs in order to replay the SQL statement. Examples of such actions are specifying data at execution time, performing compound SQL, or using array input.
- The application is not a stored procedure.

- The application is not running in a Federated environment.
- Two-phase commit is used, if transactions are dependent on the success of previous transactions. When a failure occurs during a commit operation, the client has no information about whether work was committed or rolled back at the server. If each transaction is dependent on the success of the previous transaction, use two-phase commit. Two-phase commit requires the use of XA support.

Chapter 18. XA support for a Sysplex in non-Java clients

IBM data server clients and non-Java data server drivers that have a DB2 Connect license can directly access a DB2 for z/OS Sysplex and use native XA support without going through a middle-tier DB2 Connect server.

This type of client-side XA support is only available for transaction managers that use a single-transport processing model. In a single-transport model, a transaction, over a single transport (physical connection), is tied to a member from `xa_start` to `xa_end`. The transaction end is followed immediately by `xa_prepare(readonly)`, `xa_prepare` plus `xa_commit` or `xa_rollback`, or `xa_rollback`. All of this must occur within a single application process. Examples of transaction managers that use this model include IBM TXSeries CICS, IBM WebSphere® Application Server, and Microsoft Distributed Transaction Coordinator.

Support for the single-transport processing model also includes indoubt transaction recovery where member information for each recoverable transaction is retrieved through `xa_recover`, which allows `xa_commit` or `xa_rollback` to be directed at the specified member.

You enable XA support by using the `SINGLE_PROCESS` parameter in the `xa_open` string, or by specifying settings for XA in the `db2dsdriver` configuration file.

XA support in non-Java clients has the following restrictions:

- The following transaction manager processing models are not supported:
 - Dual-transport. In this model, a transaction, over transport A, is tied to a member from `xa_start` to `xa_end`, but `xa_prepare(readonly)`, `xa_prepare` plus `xa_commit` or `xa_rollback`, or `xa_rollback` comes in over transport B, possibly from another application process. Examples of transaction managers that use this model are IBM WebSphere MQ and IBM Lotus® Domino®.
 - Multi-transport. This model involves the use of multiple transports from multiple application processes, for the same transaction.
- For XA transaction managers that use a multi-transport processing model, a middle-tier DB2 Connect server is still required.
- When XA support is enabled at the client, seamless failover is automatically disabled.

Important: DB2 for z/OS APAR PK69659 must be installed for direct XA support (needed for transaction managers such as Microsoft Distributed Transaction Coordinator). For more information, see APAR PK69659.

Enabling XA support for a Sysplex in non-Java clients

XA support for a DB2 for z/OS Sysplex can be enabled implicitly either by WLB being enabled or Microsoft Distributed Transaction Coordinator or Microsoft Component Services (COM+) being used for instance-less clients.

To explicitly enable XA support for clients that access a DB2 for z/OS Sysplex, you either specify settings in the `db2dsdriver` configuration file or use the `SINGLE_PROCESS` parameter in the `xa_open` string.

Before you begin

A DB2 Connect license is required to access the a DB2 for z/OS Sysplex.

The listed clients provide XA support for applications that access a DB2 for z/OS Sysplex:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server Driver Package
- IBM Data Server Driver for ODBC and CLI

Important: DB2 for z/OS APAR PK69659 must be installed for direct XA support (needed for transaction managers such as Microsoft Distributed Transaction Coordinator). For more information, see APAR PK69659.

About this task

This task describes how to explicitly enable XA support for IBM data server clients and non-Java data server drivers.

Restrictions

XA support is only available for transaction managers that use a single-transport processing model. For more information about this restriction, see the topic about client Sysplex limitations.

Procedure

1. For instance-based clients (IBM data server clients), specify whether XA support is on (true) or off (false) by setting the enableDirectXA parameter in the db2dsdriver configuration file, or by using the SINGLE_PROCESS parameter in the xa_open string.
2. For instance-less clients, (IBM data server drivers), XA support is enabled by default for Microsoft Distributed Transaction Coordinator or Microsoft Component Services (COM+). For all other supported transaction managers, specify whether XA support is enabled by setting the SINGLE_PROCESS keyword in the xa_open string. Settings for enableDirectXA in the db2dsdriver configuration file are not applicable to instance-less clients.

Results

If XA support is enabled, an application can run a distributed transaction over a single transport within a single application process without going through a middle-tier DB2 Connect server.

Example

Enable single-transport XA support for the database SAMPLE.

```
<database name="SAMPLE" host="v33ec065.my.domain.com" port="446">
  <!-- database-specific parameters -->
  <!--directXA is disabled by default -->
  <parameter name="enableDirectXA" value="true" />
</parameters>
</database>
```

Chapter 19. Configuring your development environment to build and run CLI and ODBC applications

You can run CLI and ODBC applications against a DB2 database server using the IBM Data Server Client, the IBM Data Server Runtime Client, or the IBM Data Server Driver for ODBC and CLI. However, to compile CLI or ODBC applications, you need the IBM Data Server Client.

Procedure

In order for a CLI application to successfully access a DB2 database:

1. Ensure the CLI/ODBC driver was installed during the DB2 client install.
2. For the IBM Data Server Client and Runtime Client only: If the database is being accessed from a remote client, catalog the database and hostname of the machine the database is on.

On Windows operating systems, you can use the CLI/ODBC Settings GUI to catalog the DB2 database.

3. Optional: Explicitly bind the 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 Windows operating systems, you can use the CLI/ODBC Settings GUI to bind the CLI/ODBC bind files to the database.

4. Optional: Change the CLI /ODBC configuration keywords by editing the `db2cli.ini` file. For information about the location of the `db2cli.ini` file, see “*db2cli.ini initialization file*” in *Call Level Interface Guide and Reference Volume 1*.

On Windows operating systems, you can use the CLI/ODBC Settings GUI to set the CLI/ODBC configuration keywords.

Results

Once you have completed steps 1 to 4, proceed to setting up your Windows CLI environment, or setting up your Linux or UNIX ODBC environment if you are running ODBC applications on Linux or UNIX.

Setting up the ODBC environment (Linux and UNIX)

This topic explains how to set up client access to DB2 databases for ODBC applications in Linux and UNIX operating systems. If your application is a CLI application, you are only required to perform the task in the *Before you begin* section to set up your environment.

Before you begin

Before setting up the ODBC environment, ensure you have set up the CLI environment.

Procedure

For ODBC applications on UNIX that need to access a DB2 database, perform the following steps:

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 (examples refer to configuration of the SAMPLE database data source):

- in the [ODBC Data Source] stanza:

```
SAMPLE=IBM DB2 ODBC DRIVER
```

which indicates that there is a data source called SAMPLE that uses the IBM DB2 ODBC DRIVER;

- in the [SAMPLE] stanza:

on AIX, for example,

```
[SAMPLE]
Driver=/u/thisuser/sql1lib/lib/libdb2.a
Description=Sample DB2 ODBC Database
```

on the Solaris operating system, for example,

```
[SAMPLE]
Driver=/u/thisuser/sql1lib/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`.

With the introduction of the 64-bit development environment, there have been a number of inconsistencies among vendors regarding the interpretation of the sizes of certain parameters. For example, the 64-bit Microsoft ODBC Driver Manager treats `SQLHANDLE` and `SQLLEN` as both 64-bits in length, whereas Data Direct Connect and open source ODBC driver managers treat `SQLHANDLE` as 64-bit, but `SQLLEN` as 32-bit. The developer must therefore pay careful attention to which version of the DB2 driver is required. Specify the appropriate DB2 driver in the data source stanza, according to the following information:

Type of application	DB2 driver to specify
32-bit CLI	libdb2.*
32-bit ODBC Driver Manager	libdb2.*
64-bit CLI	libdb2.*
64-bit ODBC Driver Manager	libdb2o.* (db2o.o for AIX)

Note: The file extension of the DB2 driver to specify depends on the operating system. The extensions are as follows:

- `.a` - AIX
- `.so` - Linux, Solaris, HP-IPF

– .sl - HP-PA

3. Ensure that the application execution environment has reference to the ODBC driver manager by including the corresponding shared library in the environment variable for the library path. The following table indicates the library name by operating system

Operating system	Environment variable	Library name
AIX	LIBPATH	libodbc.a
HP-UX, Linux, and Solaris	LD_LIBRARY_PATH	libodbc.so

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.

Sample build scripts and configurations for the unixODBC driver manager

The unixODBC driver manager is an open source ODBC driver manager for use on the Linux and UNIX platforms. The unixODBC driver manager is supported for use with ODBC applications on supported DB2 platforms. There are several build scripts and configurations that you can use with the unixODBC driver manager.

Support statement

If you experience problems with the combination of the unixODBC driver manager and the DB2 ODBC driver after they are properly installed and configured, you can contact the DB2 Service (<http://www.ibm.com/software/data/db2/udb/support>) for assistance in diagnosing the problem. If the source of the problem lies with the unixODBC driver manager, you can:

- Purchase a service contract for technical support from Easysoft, a commercial sponsor of the unixODBC (<http://www.easysoft.com>) driver manager.
- Participate in any open source support channels at <http://www.unixodbc.org>.

Sample build scripts

The following examples are sample build scripts for setting up your environment to use the unixODBC driver manager.

AIX

```
#!/bin/sh

echo "Unzipping and extracting"
gzip -d unixODBC-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar

cd unixODBC-2.2.11
```

```

#Comment this out if not AIX
export CC=xlc_r
export CCC=xlC_r

echo "Configuring, compiling and installing"
configure --prefix=$HOME --enable-gui=no --enable-drivers=no
make
make install

echo "Setting ini env vars."
export ODBCHOME=~/.etc
export ODBCINI=~/.odbc.ini

#Comment this out if not AIX
echo "Extracting unixODBC libraries"
cd ~/lib
ar -x libodbc.a
ar -x libodbcinst.a
ar -x libodbccr.a

echo "\n***Still need to set up your ini files"

```

UNIX (other than the AIX platform)

```

#!/bin/sh

echo "Unzipping and extracting"
gzip -d unixODBC-2.2.11.tar.gz
tar xf unixODBC-2.2.11.tar

cd unixODBC-2.2.11

echo "Configuring, compiling and installing"
configure --prefix=$HOME --enable-gui=no --enable-drivers=no
make
make install

echo "Setting ini env vars."
export ODBCHOME=~/.etc
export ODBCINI=~/.odbc.ini

echo "\n***Still need to set up your ini files"

```

Sample INI file configurations

The following examples are sample user and system INI files for using the unixODBC driver manager.

User INI file (odbc.ini)

The odbc.ini file is in the bin subdirectory of the unixODBC driver manager installation path.

```

[DEFAULT]
Driver = DB2

[SAMPLE]
DESCRIPTION = Connection to DB2
DRIVER = DB2

```

System INI file (odbcinst.ini)

The odbcinst.ini file is in the bin subdirectory of the unixODBC driver manager installation path. The **Driver** entry for each database section must point to the location where DB2 libraries are located.

There are multiple library files that are associated with DB2 ODBC driver. You must specify a DB2 library based on your application environment.

- For a 32-bit unixODBC driver manager on the AIX platform, you must specify the `libdb2.a` library for the **Driver** entry.
- For a 32-bit unixODBC driver manager on the Linux and UNIX platform other than AIX, you must specify the `libdb2.so` library for the **Driver** entry.
- For a 64-bit unixODBC driver manager on the AIX platform, you must specify the `db2o.o` library for the **Driver** entry.
- For a 64-bit unixODBC driver manager on the Linux and UNIX platform other than AIX, you must specify the `libdb2o.so` library for the **Driver** entry.

The following sample system INI file has the ODBC trace enabled, with the trace log file set to `trc.log`.

```
[DEFAULT]
Description = Default Driver
Driver = /u/db2inst1/sqllib/lib/db2o.o
fileusage=1
dontdlclose=1

[DB2]
Description = DB2 Driver
Driver = /u/db2inst1/sqllib/lib/db2o.o
fileusage=1
dontdlclose=1

[ODBC]
Trace = yes
Tracefile = /u/user/trc.log
```

Note:

- If you encounter problems when closing the driver manager, such as during `SQLDisconnect()`, set the value `dontdlclose=1` in the `odbcinst.ini` file, as shown in the sample system INI file.
- The 64-bit unixODBC driver manager version 2.3.0 and later treats the `SQLHANDLE` and `SQLLEN` value as 64-bits in length. The 64-bit unixODBC driver manager versions before the version 2.3.0 treats the `SQLHANDLE` value as 64-bit in length, but the `SQLLEN` value as 32-bit in length.

Setting up the Windows CLI environment

On Windows platforms, CLI driver must be registered with the Windows ODBC Data Source Administrator (`odbcad32.exe`), before it can be used by an ODBC application.

Before you begin

Before setting up the Windows CLI environment, ensure that the CLI environment is set up.

About this task

The CLI driver implements both CLI application programming interface (API) and the ODBC API. In Windows environment, CLI driver must be registered with the Windows ODBC Data Source Administrator (`odbcad32.exe`) before it can be used by an ODBC application. When using the ODBC Data Source Administrator on Windows 64-bit platforms, by default ODBC data sources can be configured only for 64-bit applications. ODBC data sources for 32-bit applications should be

configured by using the Microsoft 32-bit ODBC Data Source Administrator (32-bit `odbcad32.exe`) that is included with the Windows 64-bit operating system.

- To set up Data Sources for 32-bit applications, you must use `%WINDIR%\SysWOW64\odbcad32.exe`.
- To set up Data Sources for 64-bit applications, you must use `%WINDIR%\System32\odbcad32.exe`.

Procedure

Before CLI and ODBC applications can successfully access a DB2 database from a Windows client, perform the listed steps on the client system:

1. Verify that the Microsoft ODBC Driver Manager and the CLI/ODBC driver are installed. On Windows operating systems, both drivers are installed with the DB2 database products. If a newer version of the Microsoft ODBC Driver Manager is already installed or you manually cleared the option to install it, the Microsoft ODBC Driver Manager is not installed. To verify that both drivers are installed perform the listed actions:
 - a. Double-click the Microsoft ODBC data sources icon in the Control Panel, or run the **odbcad32.exe** command from the command line.
 - b. Click the **Drivers** tab.
 - c. Verify that the IBM DB2 ODBC DRIVER - *DB2_Copy_Name* is shown in the list. *DB2_Copy_Name* is the DB2 copy name that you want to use.

If either the Microsoft ODBC Driver Manager or the IBM Data Server Driver for ODBC and CLI is not installed, then rerun the DB2 installation 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 can be downloaded from www.microsoft.com.

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 any of these methods to add the data source:
 - Use the **db2cli** command with the **registerdsn** parameter:
 - Issue the **db2cli** command for each data source that you want to add as follows:

```
db2cli registerdsn -add data-source-name
```
 - Use the Microsoft ODBC Administration tool, which you can access from the Control Panel or by running the **odbcad32.exe** command from the command line:
 - a. The list of user data sources is shown by default. If you want to add a system data source click the **System DSN** button, or the **System DSN** tab (depending on the platform).
 - b. Click **Add**.
 - c. Double-click the IBM DB2 ODBC DRIVER - *DB2_Copy_Name* in the list. *DB2_Copy_Name* is the DB2 copy name that you want to use.
 - d. Select the DB2 database to add and click **OK**.
 - Use the **CATALOG** command to register the DB2 database with the ODBC driver manager as a data source. For example:

```
CATALOG [ user | system ] ODBC DATA SOURCE
```

Using this command, an administrator can create a command line processor script to register the required databases. This script can then be run on all computers that require access to DB2 databases through ODBC.

Results

After configuring the Windows CLI environment, you can now access DB2 data source from Windows ODBC applications.

Selecting a different DB2 copy for your Windows CLI application

By default, CLI applications running on Windows systems make use of the default DB2 copy. However, applications can use any DB2 copy that is installed on the system.

Before you begin

Ensure your Windows CLI environment is set up.

Procedure

The following methods allow CLI applications to successfully access a different DB2 copy on Windows operating systems:

- Using the DB2 command window from the **Start > Programs > IBM DB2 > DB2_Copy_Name > Command Line Tools > DB2 Command Window**: the command window is already set up with the correct environment variables for the particular DB2 copy chosen.

- Using `db2envvar.bat` from a command window:

1. Open a command window.
2. Run the `db2envvar.bat` file using the fully qualified path for the DB2 copy that you want the application to use:

```
DB2_Copy_install_dir\bin\db2envvar.bat
```

3. Run the CLI application from the same command window.

This will set up all the environment variables for the selected DB2 copy in the command window where the `db2envvar.bat` was run. Once the command window has been closed and a new one opened, the CLI application will run against the default DB2 Copy unless the `db2envvar.bat` for another DB2 copy is run again.

- Using the **db2SelectDB2Copy** API: For applications that are dynamically linked, you can call this API before loading any DB2 DLLs within your application process. This API sets up the required environment for your application to use the DB2 copy that you want to use. The **/delayload** linking option can be used to delay the loading of any DB2 DLL. For example, if your CLI application links `db2api.lib`, then you must use the **/delayload** option of your linker to delay the load `db2app.dll`:

```
c1 -Zi -MDd -Tp App.C /link /DELAY:nobind /DELAYLOAD:db2app.dll  
advapi32.lib psapi.lib db2api.lib delayimp.lib
```

To use the API, you will need to include `db2ApiInstall.h`, which will force your application to statically link in `db2ApiInstall.lib`.

- Using `LoadLibraryEx`: Instead of using `LoadLibrary`, you can call `LoadLibraryEx` with the `LOAD_WITH_ALTERED_SEARCH_PATH` parameter to load the `db2app.dll` that corresponds to the version of the DB2 copy you want to use. For example:

```
HMODULE hLib = LoadLibraryEx("c:\\sql\\bin\\db2app.dll",
    NULL, LOAD_WITH_ALTERED_SEARCH_PATH);
```

CLI bind files and package names

CLI packages are automatically bound to databases when the databases are created or upgraded, or a fix pack is applied to either the client or the server. If a user has intentionally dropped a package, then you must rebind `db2cli.lst`.

Rebind `db2cli.lst` by issuing the following command:

Linux and UNIX

```
db2 bind BNDPATH/db2cli.lst blocking all grant public
```

Windows

```
db2 bind "%DB2PATH%\bnd\@db2cli.lst" blocking all grant public
```

The `db2cli.lst` file contains the names of the required bind files for CLI to connect to DB2 servers on Linux, UNIX, and Windows (`db2clipk.bnd` and `db2clist.bnd`).

For host and IBM Power Systems servers use one of `ddcsvm.lst`, `ddcsmvs.lst`, `ddcsvse.lst`, or `ddcs400.lst` bind list files.

Warnings that are generated when binding CLI packages (such as `db2clist.bnd` or `db2cli.lst`) to workstation or host servers are expected. This is because DB2 database systems use generic bind files, but the bind file packages for CLI packages contain sections that apply to specific platforms. Therefore, a DB2 database system might generate warnings during the binding against a server, when it encounters a platform-specific section that does not apply to the server.

The following message is an example of a warning that can be ignored which might occur when binding a CLI package (such as `db2clist.bnd` or `db2cli.lst`) to a workstation server:

```
LINE    MESSAGES FOR db2clist.bnd
-----
235    SQL0440N  No authorized routine named "POSSTR" of type
        "FUNCTION" having compatible arguments was found.
        SQLSTATE=42884
```

Table 39. CLI bind files and package names

Bind file name	Package name	Needed by DB2 servers on Linux, UNIX, and Windows	Needed by host servers	Description
db2cli.pk.bnd	SYSSHxyy	Yes	Yes	dynamic placeholders - small package WITH HOLD
	SYSSNxyy	Yes	Yes	dynamic placeholders - small Package NOT WITH HOLD
	SYSLHxyy	Yes	Yes	dynamic placeholders - large package WITH HOLD
	SYSLNxyy	Yes	Yes	dynamic placeholders - large package NOT WITH HOLD
db2clist.bnd	SYSSTAT	Yes	Yes	common static CLI functions
db2schema.bnd	SQLL9vyy	Yes	No	catalog function support

Note:

- 'S' represents a small package and 'L' represents a large package
- 'H' represents WITH HOLD, and 'N' represents NOT WITH HOLD.
- 'v' represents the DB2 server version: for example, E=Version 8, F=Version 9
- 'x' is the isolation level: 0=NC, 1=UR, 2=CS, 3=RS, 4=RR
- 'yy' is the package iteration 00 through FF
- 'zz' is unique for each platform

For example, for the dynamic packages:

- SYSSN100 A small package (65 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 (385 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.

db2schema.bnd bind file

The db2schema.bnd bind file is automatically bound when the database is created or upgraded, or a fix pack is applied on DB2 servers on Linux, UNIX, and Windows, and exists only on these types of servers. This bind file is located at the

server and should be bound manually (from the server), if the package was intentionally dropped by a user or if an SQL1088W (+1088) warning is received after database creation or upgrade.

Only the most recent version of this package is needed.

If the package is missing, it must be rebound locally on the server. Do not bind this package against remote servers (for example, against a host database). The bind file is found in the sqllib/bnd directory of the instance home directory, and is rebound with the following command:

```
bind db2schema.bnd blocking all grant public
```

If an SQL1088W warning was received after database creation or upgrade, and the db2schema.bnd package is missing, increase the **applheapsz** database configuration parameter to 128 or greater, and attempt to rebound. No errors should be reported during binding.

Bind option limitations for CLI packages

Some bind options might not take effect when binding CLI packages with any of the following list files: db2cli.lst, ddcsmvs.lst, ddcs400.lst, ddcsvm.lst, or ddcsvse.lst. Because CLI packages are used by CLI, ODBC, JDBC, OLE DB, .NET, and ADO applications, any changes made to the CLI packages affect all applications of these types.

Some bind options might not take effect when binding CLI packages with any of the following list files: db2cli.lst, ddcsmvs.lst, ddcs400.lst, ddcsvm.lst, or ddcsvse.lst. Because CLI packages are used by CLI, ODBC, JDBC, OLE DB, .NET, and ADO applications, any changes made to the CLI packages affect all applications of these types. Only a subset of bind options are therefore supported by default when binding CLI packages. The supported options are: ACTION, COLLECTION, CLIPKG, OWNER, and REPLVER. All other bind options that impact CLI packages are ignored.

To create CLI packages with bind options that are not supported by default, specify the COLLECTION bind option with a collection ID that is different from the default collection ID, NULLID. Any bind options specified are then accepted. For example, to create CLI packages with the KEEP DYNAMIC YES bind option, which is not supported by default, issue the following command:

```
db2 bind @db2cli.lst collection newcolid keepdynamic yes
```

In order for CLI/ODBC applications to access the CLI packages created in the new collection, set the CurrentPackageSet CLI/ODBC keyword in the db2cli.ini initialization file to the new collection ID.

To overwrite CLI packages that already exist under a particular collection ID, perform either of the following actions:

- Drop the existing CLI package before issuing the bind command for this collection ID.
- Specify the ACTION REPLACE bind option when issuing the bind command.

Chapter 20. Building CLI applications

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 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 `.sql` extension, then the `embprep` script is called to precompile the program, producing a program file with a `.c` extension.

About this task

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
```

Procedure

- **Building and Running Embedded SQL Applications** There are three ways to build the embedded SQL application, `dbusemx`, from the source file `dbusemx.sql`:

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
```

AIX CLI application compile and link options

The compile and link options in this topic 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 options:

xlc The IBM C compiler.

\$EXTRA_CFLAG

Contains the value "-q64" for 64-bit environments; otherwise, 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 the value "-q64" for 64-bit environments; otherwise, 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.

HP-UX CLI application compile and link options

The compile and link options in this topic 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 options:

cc Use the C compiler.

\$EXTRA_CFLAG

If the HP-UX platform is IA64 and 64-bit support is enabled, this flag contains the value **+DD64**; if 32-bit support is enabled, it contains the value **+DD32**.

- +DD64** Must be used to generate 64-bit code for HP-UX on IA64.
- +DD32** Must be used to generate 32-bit code for HP-UX on IA64.
- Ae** Enables HP ANSI extended mode.
- I\$DB2PATH/include**
Specify the location of the DB2 include files. For example:
\$HOME/sql1lib/include
- mt** Enable multi-thread support. This option is required when you are compiling C/C++ applications.
- c** Perform compile only; no link. Compile and link are separate steps.

Link options:

cc Use the compiler as a front end for the linker.

\$EXTRA_CFLAG

If the HP-UX platform is IA64 and 64-bit support is enabled, this flag contains the value **+DD64**; if 32-bit support is enabled, it contains the value **+DD32**.

+DD64 Must be used to generate 64-bit code for HP-UX on IA64.

+DD32 Must be used to generate 32-bit code for HP-UX on IA64.

-o \$1 Specify the executable program.

\$1.o Specify the object file.

utilcli.o

Include the utility object file for error checking.

\$EXTRA_LFLAG

Specify the runtime path. If set, for 32-bit it contains the value **-Wl,+b\$HOME/sql1lib/lib32**, and for 64-bit: **-Wl,+b\$HOME/sql1lib/lib64**. If not set, it contains no value.

-L\$DB2PATH/\$LIB

Specify the location of the DB2 runtime shared libraries. For 32-bit: **\$HOME/sql1lib/lib32**; for 64-bit: **\$HOME/sql1lib/lib64**.

-ldb2 Link with the database manager library.

-mt Enable multi-thread support. This option is required when you are linking to C/C++ applications.

-lunwind

Link with the HP-UX unwind library to support DB2 problem determination features. This option is required.

Refer to your compiler documentation for additional compiler options.

Linux CLI application compile and link options

The compile and link options in this topic are recommended by DB2 for building CLI applications with the GNU/Linux gcc compiler. They are demonstrated in the `sql1lib/samples/cli/bldapp` build script.

Compile options:

gcc The C compiler.

\$EXTRA_C_FLAGS

Consists of one of the listed flags:

- -m31 on Linux for zSeries® only, to build a 32-bit library;
- -m32 on Linux for x86, x64 and POWER®, to build a 32-bit library;
- -m64 on Linux for zSeries, POWER, x64, to build a 64-bit library; or
- No value on Linux for IA64, to build a 64-bit library.

-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.

\$EXTRA_C_FLAGS

Consists of one of the listed flags:

- -m31 on Linux for zSeries only, to build a 32-bit library;
- -m32 on Linux for x86, x64 and POWER, to build a 32-bit library;
- -m64 on Linux for zSeries, POWER, x64, to build a 64-bit library; or
- No value on Linux for IA64, to build a 64-bit library.

-o \$1 Specify the executable.

\$1.o Include the program object file.

utilcli.o

Include the utility object file for error checking.

\$EXTRA_LFLAG

For 32-bit it contains the value "-Wl,-rpath,\$DB2PATH/lib32", and for 64-bit it contains the value "-Wl,-rpath,\$DB2PATH/lib64".

-L\$DB2PATH/\$LIB

Specify the location of the DB2 static and shared libraries at link-time. For example, for 32-bit: \$HOME/sqllib/lib32, and for 64-bit: \$HOME/sqllib/lib64.

-ldb2 Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

Solaris CLI application compile and link options

The compile and link options in this topic are recommended by DB2 for building CLI applications with the Solaris 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 as follows:

- "v8plusa" for 32-bit applications on Solaris SPARC

- "v9" for 64-bit applications on Solaris SPARC
- "sse2" for 32-bit applications on Solaris x64
- "amd64" for 64-bit applications on Solaris x64

-I\$DB2PATH/include

Specify the location of the DB2 include files. For example:
\$HOME/sql1lib/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.

- 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, for 32-bit: \$HOME/sql1lib/lib32, and for 64-bit: \$HOME/sql1lib/lib64.

\$EXTRA_LFLAG

Specify the location of the DB2 shared libraries at run time. For 32-bit it contains the value "-R\$DB2PATH/lib32", and for 64-bit it contains the value "-R\$DB2PATH/lib64".

- ldb2** Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

Building CLI multi-connection applications on UNIX

DB2 for Linux, UNIX, and Windows provides build scripts for compiling and linking CLI programs. These are located in the sql1lib/samples/cli directory, along with sample programs that can be built with these files.

About this task

The build file, bldmc, contains the commands to build a DB2 multi-connection program, requiring two databases. The compile and link options are the same as those used in bldapp.

The first parameter, \$1, specifies the name of your source file. The second parameter, \$2, specifies the name of the first database to which you want to

connect. The third parameter, \$3, specifies the second database to which you want to connect. These are all required parameters.

Note: The makefile hardcodes default values of "sample" and "sample2" for the database names (\$2 and \$3) so if you are using the makefile, and accept these defaults, you only have to specify the program name (the \$1 parameter). If you are using the `blmcc` script, you must specify all three parameters.

Optional parameters are not required for a local connection, but are required for connecting to a server from a remote client. These are: \$4 and \$5 to specify the user ID and password for the first database; and \$6 and \$7 to specify the user ID and password for the second database.

For the multi-connection sample program, `dbmconx`, you require two databases. If the `sample` database is not yet created, you can create it by entering **db2samp1** on the command line. The second database, here called `sample2`, can be created with one of the following commands:

Procedure

- If creating the database locally:

```
db2 create db sample2
```

- If creating the database remotely:

```
db2 attach to node_name
db2 create db sample2
db2 detach
db2 catalog db sample2 as sample2 at node node_name
```

where *node_name* is the database partition where the database resides.

- Multi-connection also requires that the TCP/IP listener is running. To ensure that the TCP/IP listener is running, follow the listed steps:

1. Set the environment variable **DB2COMM** to TCP/IP as follows:

```
db2set DB2COMM=TCPIP
```

2. Update the database manager configuration file with the TCP/IP service name as specified in the services file:

```
db2 update dbm cfg using SVCENAME TCP/IP_service_name
```

Each instance has a TCP/IP service name listed in the services file. Ask your system administrator if you cannot locate it or do not have the file permission to read the services file. On UNIX and Linux systems, the services file is located in: `/etc/services`

3. Stop and restart the database manager in order for these changes to take effect:

```
db2stop
db2start
```

The `dbmconx` program consists of five files:

dbmconx.c

Main source file for connecting to both databases.

dbmconx1.sqc

Source file for creating a package bound to the first database.

dbmconx1.h

Header file for dbmconx1.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the first database.

dbmconx2.sqc

Source file for creating a package bound to the second database.

dbmconx2.h

Header file for dbmconx2.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the second database.

- To build the multi-connection sample program, dbmconx, enter:

```
bldmc dbmconx sample sample2
```

The result is an executable file, dbmconx.

- To run the executable file, enter the executable name:

```
dbmconx
```

The program demonstrates a two-phase commit to two databases.

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.

About this task

The batch file `bldapp.bat` contains the commands to build a 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.

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`:

Procedure

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:

- a. If accessing the sample database on the same instance, simply enter the executable name:
`dbusemx`
- b. If accessing another database on the same instance, enter the executable name and the database name:
`dbusemx database`
- c. 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`

Windows CLI application compile and link options

The compile and link options in this topic 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 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 and 64-bit applications, or `ec1`, 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.
- W2** Set warning level.
- DWIN32** Compiler option necessary for Windows operating systems.
- J** Compiler option that changes the default char type from signed char to unsigned char. The char type is zero-extended when widened to an int type. If a char type is explicitly declared as signed char then the `-J` option has no effect.

Link options:

- link** Use the 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.

db2api.lib
Link with the DB2 API library.

/delayload:db2app.d11
Used to ensure that db2app.d11 is not loaded until the first call to a DB2 API. This is required when using the db2SelectDB2Copy API.

db2ApiInstall.lib
Library to statically link in your application if you need to select a particular DB2 copy that is installed on the computer using the db2SelectDB2Copy API. Note: to use this functionality, you need to either dynamically load db2app.d11 or use the /delayload:db2app.d11 option of your compiler and call the db2SelectDB2Copy API before running any other DB2 API's.

Refer to your compiler documentation for additional compiler options.

Building CLI multi-connection applications on Windows

DB2 provides batch files for compiling and linking CLI programs. These are located in the sql1lib\samples\cli directory, along with sample programs that can be built with these files.

About this task

The batch file, bldmc.bat, contains the commands to build a DB2 multi-connection program requiring two databases. The compile and link options are the same as those used in bldapp.bat.

The first parameter, %1, specifies the name of your source file. The second parameter, %2, specifies the name of the first database to which you want to connect. The third parameter, %3, specifies the second database to which you want to connect. These are all required parameters.

Note: The makefile hardcodes default values of "sample" and "sample2" for the database names (%2 and %3) so if you are using the makefile, and accept these defaults, you only have to specify the program name (the %1 parameter). If you are using the bldmc.bat file, you must specify all three parameters.

Optional parameters are not required for a local connection, but are required for connecting to a server from a remote client. These are: %4 and %5 to specify the user ID and password for the first database; and %6 and %7 to specify the user ID and password for the second database.

For the multi-connection sample program, dbmconx, you require two databases. If the sample database is not yet created, you can create it by entering db2sample on the command line. The second database, here called sample2, can be created with one of the following commands:

Procedure

- If creating the database locally:
`db2 create db sample2`
- If creating the database remotely:
`db2 attach to <node_name>`
`db2 create db sample2`
`db2 detach`
`db2 catalog db sample2 as sample2 at node <node_name>`

where <node_name> is the database partition where the database resides.

- Multi-connection also requires that the TCP/IP listener is running. To ensure that the TCP/IP listener is running, follow the listed steps:

1. Set the environment variable DB2COMM to TCP/IP as follows:

```
db2set DB2COMM=TCPIP
```

2. Update the database manager configuration file with the TCP/IP service name as specified in the services file:

```
db2 update dbm cfg using SVCENAME <TCP/IP service name>
```

Each instance has a TCP/IP service name listed in the services file. Ask your system administrator if you cannot locate it or do not have the file permission to read the services file.

3. Stop and restart the database manager in order for these changes to take effect:

```
db2stop  
db2start
```

The dbmconx program consists of five files:

dbmconx.c

Main source file for connecting to both databases.

dbmconx1.sqc

Source file for creating a package bound to the first database.

dbmconx1.h

Header file for dbmconx1.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the first database.

dbmconx2.sqc

Source file for creating a package bound to the second database.

dbmconx2.h

Header file for dbmconx2.sqc included in dbmconx.sqc for accessing the SQL statements for creating and dropping a table to be bound to the second database.

- To build the multi-connection sample program, dbmconx, enter:

```
blc dbmconx sample sample2
```

The result is an executable file, dbmconx.

- To run the executable file, enter the executable name:

```
dbmconx
```

The program demonstrates a two-phase commit to two databases.

Building CLI applications with configuration files

The configuration file, `cli.icc`, in `sqllib/samples/cli` allows you to build CLI programs.

Procedure

To use the configuration file to build the CLI sample program `tbinfo` from the source file `tbinfo.c`:

1. Set the CLI environment variable:
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:

```
export CLI=tbinfo
```

```
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
```

Results

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`:

- If connecting to the sample database on the same instance, enter:

```
embprep dbusemx
```

- If connecting to another database on the same instance, also enter the database name:

```
embprep dbusemx database
```

- 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:

- If accessing the sample database on the same instance, simply enter the executable name:

```
dbusemx
```

- If accessing another database on the same instance, enter the executable name and the database name:

```
dbusemx database
```

- 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
```

Building CLI stored procedures with configuration files

The configuration file, `clis.icc`, in `sqllib/samples/cli`, allows you to build CLI stored procedures.

Procedure

To use the configuration file to build the CLI stored procedure `spserver` from the source file `spserver.c`:

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++.

4. 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.

Chapter 21. Building CLI routines

Building CLI routines on UNIX

DB2 for Linux, UNIX, and Windows provides build scripts for compiling and linking DB2 Call Level Interface (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 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 `spscreate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `spscreate.db2` scripts individually.

3. Then, unless this is the first time the shared library was built, stop and restart the database to allow the new version of the shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Results

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.

AIX CLI routine compile and link options

The compile and link options in this topic 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 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 the value "-q64" for 64-bit environments; otherwise, 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 the value "-q64" for 64-bit environments; otherwise, 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.

HP-UX CLI routine compile and link options

The compile and link options in this topic 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 options:

cc The C compiler.

\$EXTRA_CFLAG

If the HP-UX platform is IA64 and 64-bit support is enabled, this flag contains the value **+DD64**; if 32-bit support is enabled, it contains the value **+DD32**.

+DD64 Must be used to generate 64-bit code for HP-UX on IA64.

+DD32 Must be used to generate 32-bit code for HP-UX on IA64.

+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`).

-mt Enable multi-thread support. This option is required when you are compiling C/C++ routines.

-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.

\$EXTRA_LFLAG

Specify the runtime path. If set, for 32-bit it contains the value `+b$HOME/sqllib/lib32`, and for 64-bit: `+b$HOME/sqllib/lib64`. If not set, it contains no value.

-L\$DB2PATH/\$LIB

Specify the location of the DB2 runtime shared libraries. For 32-bit: `$HOME/sqllib/lib32`; for 64-bit: `$HOME/sqllib/lib64`.

-ldb2 Link with the DB2 library.

-lpthread

Link with the POSIX thread library.

-lunwind

Link with the HP-UX unwind library to support DB2 problem determination features. This option is required.

Refer to your compiler documentation for additional compiler options.

Linux CLI routine compile and link options

The compile and link options in this topic are recommended by DB2 for building CLI routines with the GNU/Linux gcc compiler. They are demonstrated in the `sql/lib/samples/cli/bldrtn` build script.

Compile options:

gcc The C compiler.

\$EXTRA_C_FLAGS

Consists of one of the listed flags:

- `-m31` on Linux for zSeries only, to build a 32-bit library;
- `-m32` on Linux for x86, x64 and POWER, to build a 32-bit library;
- `-m64` on Linux for zSeries, POWER, x64, to build a 64-bit library; or
- No value on Linux for IA64, to build a 64-bit library.

-fpic Allows position independent code.

-I\$DB2PATH/include

Specify the location of the DB2 include files. For example:
`$HOME/sql/lib/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).

Link options:

gcc Use the compiler as a front end for the linker.

\$EXTRA_C_FLAGS

Consists of one of the listed flags:

- `-m31` on Linux for zSeries only, to build a 32-bit library;
- `-m32` on Linux for x86, x64 and POWER, to build a 32-bit library;
- `-m64` on Linux for zSeries, POWER, x64, to build a 64-bit library; or
- No value on Linux for IA64, to build a 64-bit library.

-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.

\$EXTRA_LFLAG

Specify the location of the DB2 shared libraries at run time. For 32-bit it contains the value `"-Wl,-rpath,$DB2PATH/lib32"`. For 64-bit it contains the value `"-Wl,-rpath,$DB2PATH/lib64"`.

-L\$DB2PATH/\$LIB

Specify the location of the DB2 static and shared libraries at link-time. For example, for 32-bit: \$HOME/sql1lib/lib32, and for 64-bit: \$HOME/sql1lib/lib64.

-ldb2 Link with the DB2 library.

-pthread

Link with the POSIX thread library.

Refer to your compiler documentation for additional compiler options.

Solaris CLI routine compile and link options

The compile and link options in this topic are recommended by DB2 for building CLI routines with the Solaris C compiler. They are demonstrated in the `sql1lib/samples/cli/bldrtn` build script.

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 as follows:

- "v8plusa" for 32-bit applications on Solaris SPARC
- "v9" for 64-bit applications on Solaris SPARC
- "sse2" for 32-bit applications on Solaris x64
- "amd64" for 64-bit applications on Solaris x64

-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/sql1lib/include.

-c Perform compile only; no link. Compile and link are separate 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.

-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, for 32-bit: `$HOME/sql1lib/lib32`, and for 64-bit: `$HOME/sql1lib/lib64`.

\$EXTRA_LFLAG

Specify the location of the DB2 shared libraries at run time. For 32-bit it contains the value `"-R$DB2PATH/lib32"`, and for 64-bit it contains the value `"-R$DB2PATH/lib64"`.

-ldb2 Link with the DB2 library.

Refer to your compiler documentation for additional compiler options.

Building CLI routines on Windows

DB2 for Linux, UNIX, and Windows provides batch files for compiling and linking CLI programs.

These are located in the `sql1ib\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 `sql1ib\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 `screate.db2`, and finally disconnects from the database. You can also call the `spdrop.db2` and `screate.db2` scripts individually.

3. Then, unless this is the first time the shared library was built, stop and restart the database to allow the new version of the shared library to be recognized. If necessary, set the file mode for the shared library so the DB2 instance can access it.

Results

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, `bldapp`.

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.

Windows CLI routine compile and link options

The compile and link options in this topic 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 options:

%BLDCOMP%

Variable for the compiler. The default is `c1`, the Microsoft Visual C++ compiler. It can be also set to `ic1`, the Intel C++ Compiler for 32-bit and 64-bit applications, or `ec1`, 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.

-MD Link using `MSVCRT.LIB`

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.

db2api.lib

Link with the DB2 API library.

-def:%1.def

Use the module definition file.

/delayload:db2app.dll

Used to ensure that db2app.dll is not loaded until the first call to a DB2 API. This is required when using the db2SelectDB2Copy API.

db2ApiInstall.lib

Library to statically link in your application if you need to select a particular DB2 copy that is installed on the computer using the db2SelectDB2Copy API. Note: to use this functionality, you need to either dynamically load db2app.dll or use the /delayload:db2app.dll option of your compiler and call the db2SelectDB2Copy API before running any other DB2 API's.

Refer to your compiler documentation for additional compiler options.

Appendix A. Overview of the DB2 technical information

DB2 technical information is available in multiple formats that can be accessed in multiple ways.

DB2 technical information is available through the following tools and methods:

- DB2 Information Center
 - Topics (Task, concept and reference topics)
 - Sample programs
 - Tutorials
- DB2 books
 - PDF files (downloadable)
 - PDF files (from the DB2 PDF DVD)
 - printed books
- Command-line help
 - Command help
 - Message help

Note: The DB2 Information Center topics are updated more frequently than either the PDF or the hardcopy books. To get the most current information, install the documentation updates as they become available, or refer to the DB2 Information Center at ibm.com.

You can access additional DB2 technical information such as technotes, white papers, and IBM Redbooks® publications online at [ibm.com](http://www.ibm.com). Access the DB2 Information Management software library site at <http://www.ibm.com/software/data/sw-library/>.

Documentation feedback

We value your feedback on the DB2 documentation. If you have suggestions for how to improve the DB2 documentation, send an email to db2docs@ca.ibm.com. The DB2 documentation team reads all of your feedback, but cannot respond to you directly. Provide specific examples wherever possible so that we can better understand your concerns. If you are providing feedback on a specific topic or help file, include the topic title and URL.

Do not use this email address to contact DB2 Customer Support. If you have a DB2 technical issue that the documentation does not resolve, contact your local IBM service center for assistance.

DB2 technical library in hardcopy or PDF format

The following tables describe the DB2 library available from the IBM Publications Center at www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss. English and translated DB2 Version 10.1 manuals in PDF format can be downloaded from www.ibm.com/support/docview.wss?rs=71&uid=swg27009474.

Although the tables identify books available in print, the books might not be available in your country or region.

The form number increases each time a manual is updated. Ensure that you are reading the most recent version of the manuals, as listed below.

Note: The *DB2 Information Center* is updated more frequently than either the PDF or the hard-copy books.

Table 40. DB2 technical information

Name	Form Number	Available in print	Last updated
<i>Administrative API Reference</i>	SC27-3864-00	Yes	April, 2012
<i>Administrative Routines and Views</i>	SC27-3865-01	No	January, 2013
<i>Call Level Interface Guide and Reference Volume 1</i>	SC27-3866-01	Yes	January, 2013
<i>Call Level Interface Guide and Reference Volume 2</i>	SC27-3867-01	Yes	January, 2013
<i>Command Reference</i>	SC27-3868-01	Yes	January, 2013
<i>Database Administration Concepts and Configuration Reference</i>	SC27-3871-01	Yes	January, 2013
<i>Data Movement Utilities Guide and Reference</i>	SC27-3869-01	Yes	January, 2013
<i>Database Monitoring Guide and Reference</i>	SC27-3887-01	Yes	January, 2013
<i>Data Recovery and High Availability Guide and Reference</i>	SC27-3870-01	Yes	January, 2013
<i>Database Security Guide</i>	SC27-3872-01	Yes	January, 2013
<i>DB2 Workload Management Guide and Reference</i>	SC27-3891-01	Yes	January, 2013
<i>Developing ADO.NET and OLE DB Applications</i>	SC27-3873-01	Yes	January, 2013
<i>Developing Embedded SQL Applications</i>	SC27-3874-01	Yes	January, 2013
<i>Developing Java Applications</i>	SC27-3875-01	Yes	January, 2013
<i>Developing Perl, PHP, Python, and Ruby on Rails Applications</i>	SC27-3876-00	No	April, 2012
<i>Developing RDF Applications for IBM Data Servers</i>	SC27-4462-00	Yes	January, 2013
<i>Developing User-defined Routines (SQL and External)</i>	SC27-3877-01	Yes	January, 2013
<i>Getting Started with Database Application Development</i>	GI13-2046-01	Yes	January, 2013

Table 40. DB2 technical information (continued)

Name	Form Number	Available in print	Last updated
<i>Getting Started with DB2 Installation and Administration on Linux and Windows</i>	GI13-2047-00	Yes	April, 2012
<i>Globalization Guide</i>	SC27-3878-00	Yes	April, 2012
<i>Installing DB2 Servers</i>	GC27-3884-01	Yes	January, 2013
<i>Installing IBM Data Server Clients</i>	GC27-3883-00	No	April, 2012
<i>Message Reference Volume 1</i>	SC27-3879-01	No	January, 2013
<i>Message Reference Volume 2</i>	SC27-3880-01	No	January, 2013
<i>Net Search Extender Administration and User's Guide</i>	SC27-3895-01	No	January, 2013
<i>Partitioning and Clustering Guide</i>	SC27-3882-01	Yes	January, 2013
<i>Preparation Guide for DB2 10.1 Fundamentals Exam 610</i>	SC27-4540-00	No	January, 2013
<i>Preparation Guide for DB2 10.1 DBA for Linux, UNIX, and Windows Exam 611</i>	SC27-4541-00	No	January, 2013
<i>pureXML Guide</i>	SC27-3892-01	Yes	January, 2013
<i>Spatial Extender User's Guide and Reference</i>	SC27-3894-00	No	April, 2012
<i>SQL Procedural Languages: Application Enablement and Support</i>	SC27-3896-01	Yes	January, 2013
<i>SQL Reference Volume 1</i>	SC27-3885-01	Yes	January, 2013
<i>SQL Reference Volume 2</i>	SC27-3886-01	Yes	January, 2013
<i>Text Search Guide</i>	SC27-3888-01	Yes	January, 2013
<i>Troubleshooting and Tuning Database Performance</i>	SC27-3889-01	Yes	January, 2013
<i>Upgrading to DB2 Version 10.1</i>	SC27-3881-01	Yes	January, 2013
<i>What's New for DB2 Version 10.1</i>	SC27-3890-01	Yes	January, 2013
<i>XQuery Reference</i>	SC27-3893-01	No	January, 2013

Table 41. DB2 Connect-specific technical information

Name	Form Number	Available in print	Last updated
<i>DB2 Connect Installing and Configuring DB2 Connect Personal Edition</i>	SC27-3861-00	Yes	April, 2012

Table 41. DB2 Connect-specific technical information (continued)

Name	Form Number	Available in print	Last updated
DB2 Connect Installing and Configuring DB2 Connect Servers	SC27-3862-01	Yes	January, 2013
DB2 Connect User's Guide	SC27-3863-01	Yes	January, 2013

Displaying SQL state help from the command line processor

DB2 products return an SQLSTATE value for conditions that can be the result of an SQL statement. SQLSTATE help explains the meanings of SQL states and SQL state class codes.

Procedure

To start SQL state help, open the command line processor and enter:

```
? sqlstate or ? class code
```

where *sqlstate* represents a valid five-digit SQL state and *class code* represents the first two digits of the SQL state.

For example, ? 08003 displays help for the 08003 SQL state, and ? 08 displays help for the 08 class code.

Accessing different versions of the DB2 Information Center

Documentation for other versions of DB2 products is found in separate information centers on ibm.com[®].

About this task

For DB2 Version 10.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1>.

For DB2 Version 9.8 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r8/>.

For DB2 Version 9.7 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/>.

For DB2 Version 9.5 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/>.

For DB2 Version 9.1 topics, the *DB2 Information Center* URL is <http://publib.boulder.ibm.com/infocenter/db2luw/v9/>.

For DB2 Version 8 topics, go to the *DB2 Information Center* URL at: <http://publib.boulder.ibm.com/infocenter/db2luw/v8/>.

Updating the DB2 Information Center installed on your computer or intranet server

A locally installed DB2 Information Center must be updated periodically.

Before you begin

A DB2 Version 10.1 Information Center must already be installed. For details, see the “Installing the DB2 Information Center using the DB2 Setup wizard” topic in *Installing DB2 Servers*. All prerequisites and restrictions that applied to installing the Information Center also apply to updating the Information Center.

About this task

An existing DB2 Information Center can be updated automatically or manually:

- Automatic updates update existing Information Center features and languages. One benefit of automatic updates is that the Information Center is unavailable for a shorter time compared to during a manual update. In addition, automatic updates can be set to run as part of other batch jobs that run periodically.
- Manual updates can be used to update existing Information Center features and languages. Automatic updates reduce the downtime during the update process, however you must use the manual process when you want to add features or languages. For example, a local Information Center was originally installed with both English and French languages, and now you want to also install the German language; a manual update will install German, as well as, update the existing Information Center features and languages. However, a manual update requires you to manually stop, update, and restart the Information Center. The Information Center is unavailable during the entire update process. In the automatic update process the Information Center incurs an outage to restart the Information Center after the update only.

This topic details the process for automatic updates. For manual update instructions, see the “Manually updating the DB2 Information Center installed on your computer or intranet server” topic.

Procedure

To automatically update the DB2 Information Center installed on your computer or intranet server:

1. On Linux operating systems,
 - a. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `/opt/ibm/db2ic/V10.1` directory.
 - b. Navigate from the installation directory to the `doc/bin` directory.
 - c. Run the `update-ic` script:

```
update-ic
```
2. On Windows operating systems,
 - a. Open a command window.
 - b. Navigate to the path where the Information Center is installed. By default, the DB2 Information Center is installed in the `<Program Files>\IBM\DB2 Information Center\Version 10.1` directory, where `<Program Files>` represents the location of the Program Files directory.
 - c. Navigate from the installation directory to the `doc\bin` directory.
 - d. Run the `update-ic.bat` file:

```
update-ic.bat
```

Results

The DB2 Information Center restarts automatically. If updates were available, the Information Center displays the new and updated topics. If Information Center updates were not available, a message is added to the log. The log file is located in `doc\ eclipse\ configuration` directory. The log file name is a randomly generated number. For example, `1239053440785.log`.

Manually updating the DB2 Information Center installed on your computer or intranet server

If you have installed the DB2 Information Center locally, you can obtain and install documentation updates from IBM.

About this task

Updating your locally installed *DB2 Information Center* manually requires that you:

1. Stop the *DB2 Information Center* on your computer, and restart the Information Center in stand-alone mode. Running the Information Center in stand-alone mode prevents other users on your network from accessing the Information Center, and allows you to apply updates. The Workstation version of the DB2 Information Center always runs in stand-alone mode. .
2. Use the Update feature to see what updates are available. If there are updates that you must install, you can use the Update feature to obtain and install them

Note: If your environment requires installing the *DB2 Information Center* updates on a machine that is not connected to the internet, mirror the update site to a local file system by using a machine that is connected to the internet and has the *DB2 Information Center* installed. If many users on your network will be installing the documentation updates, you can reduce the time required for individuals to perform the updates by also mirroring the update site locally and creating a proxy for the update site.

If update packages are available, use the Update feature to get the packages. However, the Update feature is only available in stand-alone mode.

3. Stop the stand-alone Information Center, and restart the *DB2 Information Center* on your computer.

Note: On Windows 2008, Windows Vista (and higher), the commands listed later in this section must be run as an administrator. To open a command prompt or graphical tool with full administrator privileges, right-click the shortcut and then select **Run as administrator**.

Procedure

To update the *DB2 Information Center* installed on your computer or intranet server:

1. Stop the *DB2 Information Center*.
 - On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Stop**.
 - On Linux, enter the following command:

```
/etc/init.d/db2icdv10 stop
```
2. Start the Information Center in stand-alone mode.
 - On Windows:
 - a. Open a command window.

- b. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the *Program_Files\IBM\DB2 Information Center\Version 10.1* directory, where *Program_Files* represents the location of the Program Files directory.
- c. Navigate from the installation directory to the `doc\bin` directory.
- d. Run the `help_start.bat` file:


```
help_start.bat
```
- On Linux:
 - a. Navigate to the path where the Information Center is installed. By default, the *DB2 Information Center* is installed in the `/opt/ibm/db2ic/V10.1` directory.
 - b. Navigate from the installation directory to the `doc/bin` directory.
 - c. Run the `help_start` script:


```
help_start
```

The systems default Web browser opens to display the stand-alone Information Center.

3. Click the **Update** button (🔄). (JavaScript must be enabled in your browser.) On the right panel of the Information Center, click **Find Updates**. A list of updates for existing documentation displays.
4. To initiate the installation process, check that the selections you want to install, then click **Install Updates**.
5. After the installation process has completed, click **Finish**.
6. Stop the stand-alone Information Center:
 - On Windows, navigate to the `doc\bin` directory within the installation directory, and run the `help_end.bat` file:


```
help_end.bat
```

Note: The `help_end` batch file contains the commands required to safely stop the processes that were started with the `help_start` batch file. Do not use `Ctrl-C` or any other method to stop `help_start.bat`.
 - On Linux, navigate to the `doc/bin` directory within the installation directory, and run the `help_end` script:


```
help_end
```

Note: The `help_end` script contains the commands required to safely stop the processes that were started with the `help_start` script. Do not use any other method to stop the `help_start` script.
7. Restart the *DB2 Information Center*.
 - On Windows, click **Start > Control Panel > Administrative Tools > Services**. Then right-click **DB2 Information Center** service and select **Start**.
 - On Linux, enter the following command:


```
/etc/init.d/db2icdv10 start
```

Results

The updated *DB2 Information Center* displays the new and updated topics.

DB2 tutorials

The DB2 tutorials help you learn about various aspects of DB2 database products. Lessons provide step-by-step instructions.

Before you begin

You can view the XHTML version of the tutorial from the Information Center at <http://publib.boulder.ibm.com/infocenter/db2luw/v10r1/>.

Some lessons use sample data or code. See the tutorial for a description of any prerequisites for its specific tasks.

DB2 tutorials

To view the tutorial, click the title.

"pureXML®" in pureXML Guide

Set up a DB2 database to store XML data and to perform basic operations with the native XML data store.

DB2 troubleshooting information

A wide variety of troubleshooting and problem determination information is available to assist you in using DB2 database products.

DB2 documentation

Troubleshooting information can be found in the *Troubleshooting and Tuning Database Performance* or the Database fundamentals section of the *DB2 Information Center*, which contains:

- Information about how to isolate and identify problems with DB2 diagnostic tools and utilities.
- Solutions to some of the most common problem.
- Advice to help solve other problems you might encounter with your DB2 database products.

IBM Support Portal

See the IBM Support Portal if you are experiencing problems and want help finding possible causes and solutions. The Technical Support site has links to the latest DB2 publications, TechNotes, Authorized Program Analysis Reports (APARs or bug fixes), fix packs, and other resources. You can search through this knowledge base to find possible solutions to your problems.

Access the IBM Support Portal at http://www.ibm.com/support/entry/portal/Overview/Software/Information_Management/DB2_for_Linux,_UNIX_and_Windows

Terms and conditions

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability: These terms and conditions are in addition to any terms of use for the IBM website.

Personal use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights: Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Trademarks: IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml

Appendix B. Notices

This information was developed for products and services offered in the U.S.A. Information about non-IBM products is based on information available at the time of first publication of this document and is subject to change.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country/region where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements, changes, or both in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to websites not owned by IBM are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information that has been exchanged, should contact:

IBM Canada Limited
U59/3600
3600 Steeles Avenue East
Markham, Ontario L3R 9Z7
CANADA

Such information may be available, subject to appropriate terms and conditions, including, in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems, and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

The following terms are trademarks or registered trademarks of other companies

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle, its affiliates, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Intel, Intel logo, Intel Inside, Intel Inside logo, Celeron, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
- Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

A

- about this book
 - Call Level Interface Guide and Reference, Volume 1 vii
- ACOS scalar function
 - vendor escape clause 182
- acrRetryInterval IBM Data Server Driver configuration keyword
 - DB2 for z/OS 233
- acrRetryInterval IBM Data Server Driver configuration parameter
 - DB2 for Linux, UNIX, and Windows 196
- alternate groups
 - DB2 for Linux, UNIX, and Windows 207
 - DB2 for z/OS 245
- alternateserverlist IBM Data Server Driver configuration parameter 196
- AltHostName CLI/ODBC configuration keyword 30
- ALHOSTNAME variable 17
- AltPort CLI/ODBC configuration keyword 31
- ALTPORT variable 17
- APD (application parameter descriptor) 139
- application parameter descriptor (APD) 139
- application programming for high availability
 - connections to DB2 for Linux, UNIX, and Windows 209
 - connections to Informix database server 224
 - direct connections to DB2 for z/OS 247
- application row descriptor (ARD) 139
- ARD (application row descriptor) 139
- arrays
 - input
 - column-wise 79
 - row-wise 79
 - output 106
- ASCII scalar function
 - vendor escape clauses 182
- ASIN scalar function
 - vendor escape clauses 182
- asynchronous function execution
 - CLI 171, 172
- ATAN scalar function
 - vendor escape clauses 182
- ATAN2 scalar function
 - vendor escape clauses 182
- Authentication CLI/ODBC keyword 31
- AUTHENTICATION variable 17
- automatic client reroute
 - alternate groups
 - DB2 for Linux, UNIX, and Windows 207
 - DB2 for z/OS 245
 - client applications 193
 - client-side
 - DB2 for Linux, UNIX, and Windows server 204
 - DB2 for z/OS server 241
 - Informix database server 221
 - configuring
 - DB2 for Linux, UNIX, and Windows 196
 - DB2 for z/OS 233
 - DB2 for z/OS 239
 - examples 199

B

- BIDI CLI/ODBC keyword 32
- BIDI variable 17
- binary large objects (BLOBs)
 - CLI applications 62
- bind files
 - package names 258
- binding
 - application variables 75, 109
 - column bindings 99, 109
 - packages
 - limitations for CLI 260
 - parameter markers
 - column-wise 79
 - details 75
 - row-wise 79
- BLOB data type
 - CLI applications 62
- bookmarks in CLI
 - deleting bulk data 120
 - details 95
 - inserting bulk data 113
 - overview 94

C

- call level interface
 - see CLI 1
- capture file 126
- case sensitivity
 - cursor name arguments 60
- catalog functions
 - overview 149
- catalogs
 - querying 149
- CEILING scalar function
 - CLI applications 182
- CHAR scalar function
 - CLI applications 182
- character strings
 - interpreting 60
 - length 60
- CICS (Customer Information Control System)
 - running applications 168
- CLI
 - AIX
 - application compiler options 262
 - application link options 262
 - routine compiler options 276
 - routine link options 276
 - applications
 - ABS scalar function 182
 - building (UNIX) 261
 - building (Windows) 267
 - building multi-connection (UNIX) 265
 - building multi-connection (Windows) 269
 - building with configuration files 271
 - DB2 Transaction Manager 166
 - initializing 55
 - issuing SQL statements 74

- CLI (*continued*)
 - applications (*continued*)
 - locators 111
 - multithreaded 177
 - terminating 131
 - XML data 71
 - array data
 - retrieving using column-wise binding 108
 - retrieving using row-wise binding 108
 - array input chaining 159
 - binding parameter markers 78
 - bookmarks
 - deleting bulk data 120
 - inserting bulk data 113
 - retrieving bulk data 105
 - retrieving data 104
 - updating bulk data 119
 - bulk data
 - deleting 120
 - inserting 113
 - retrieving 105
 - updating 119
 - compound SQL (CLI) statements
 - executing 88
 - cursors 89, 92
 - deferred prepare 87
 - deleting data 118, 120
 - descriptors
 - consistency checks 143
 - overview 139
 - drivers
 - overview 7
 - registering XA library with DTC 18
 - environment setup 251, 255
 - functions
 - executing asynchronously 171, 172
 - Unicode 162
 - handles
 - allocating 74
 - details 57
 - freeing 129
 - HP-UX
 - application compiler options 262
 - application link options 262
 - routine compiler options 277
 - IBM Data Server Driver for ODBC and CLI
 - CLI functions 35
 - configuring 11, 15, 19
 - connecting to databases 20
 - DB2 registry variables 17
 - deploying with applications 49
 - environment variables 17
 - installing 8, 9, 11
 - LDAP support 37
 - license requirements 49
 - obtaining 8
 - ODBC functions 35
 - overview 7
 - problem determination 38
 - restrictions 37
 - running database applications 35
 - XA functions 36
 - initializing 55, 56
 - Linux
 - application compiler options 263
 - application link options 263
 - routine compiler options 278
- CLI (*continued*)
 - LOB locators 64
 - long data 68
 - multithreaded applications model 176
 - overview 1
 - performance improvement 159
 - query results 97
 - routines
 - building (UNIX) 275
 - building (Windows) 280
 - building with configuration files 272
 - Solaris operating system
 - application compiler options 264
 - application link options 264
 - routine compiler options 279
 - SQL statements
 - executing 86
 - issuing 74
 - preparing 86
 - SQL/XML functions 71
 - stored procedures
 - calling 120
 - commit behavior 123
 - trusted connections 133
 - Unicode
 - functions 162
 - ODBC driver managers 163
 - Unicode applications 161
 - updating data 118
 - Windows
 - application compiler options 268
 - application link options 268
 - routine compiler options 281
 - XML data
 - changing default type 72
 - handling 71
 - inserts 117
 - retrieval 112
 - updates 117
 - XQuery expressions 71
- CLI/ODBC configuration keywords
 - AltHostName 30
 - AltPort 31
 - Authentication 31
 - BIDI 32
 - ConnectType 165
 - DiagLevel 40
 - DiagPath 41
 - FileDSN 32
 - initialization file 12
 - Instance 32
 - Interrupt 33
 - KRBPlugin 33
 - MapXMLCDefault 72
 - MapXMLDescribe 72
 - NotifyLevel 40
 - Protocol 34
 - PWDPlugin 34
 - SaveFile 34
- CLI/ODBC static profiling
 - capture file 126
 - static SQL statements 124
- CLI/ODBC/JDBC static profiling
 - capture file 126
- client affinities
 - .NET 210, 213, 225, 228
 - CLI 210, 213, 225, 228

- client affinities (*continued*)
 - IBM Data Server Driver for JDBC and SQLJ 210, 225
 - non-Java clients 210, 213, 225, 228
- client applications
 - high availability 193
 - transaction-level load balancing 193
- clients
 - enabling XA support 250
- CLOB data type
 - CLI applications 62
- column binding offsets 109
- column-wise binding 108
- columns
 - binding in CLI 99
- commits
 - CLI stored procedures 123
 - transactions
 - CLI 83
- compiler options
 - AIX
 - CLI applications 262
 - CLI routines 276
 - HP-UX
 - CLI applications 262
 - CLI routines 277
 - Linux
 - CLI applications 263
 - CLI routines 278
 - Solaris
 - CLI applications 264
 - CLI routines 279
 - Windows
 - CLI applications 268
 - CLI routines 281
- compound SQL statements
 - CLI
 - executing 88
- CONCAT scalar function
 - CLI applications 182
- concise descriptor functions 147
- connectionLevelLoadBalancing IBM Data Server Driver configuration keyword 233
- connections
 - multiple 153
 - SQLDriverConnect function 25
- ConnectType CLI/ODBC configuration keyword 165
- conversion
 - CLI applications
 - overview 59
- CONVERT scalar function 182
- coordinated transactions
 - distributed 165
 - establishing 166
- COS scalar function
 - CLI applications 182
- COT scalar function
 - CLI applications 182
- CURDATE scalar function 182
- cursors
 - call level interface (CLI)
 - bookmarks 95
 - details 89
 - selection 92
 - dynamic scrollable 89
 - holding across rollbacks 153
 - scrollable
 - retrieving data in CLI 102

- CURTIME scalar function 182
- Customer Information Control System (CICS)
 - running applications 168

D

- data
 - deleting
 - CLI applications 120
 - inserting
 - CLI applications 113
 - XML 117
 - retrieving
 - CLI 110
- data sources
 - connecting to
 - SQLDriverConnect function 25
- DATABASE scalar function 182
- DAYNAME scalar function
 - CLI applications 182
- DAYOFMONTH scalar function 182
- DAYOFWEEK scalar function
 - CLI applications 182
- DAYOFWEEK_ISO scalar function
 - CLI applications 182
- DAYOFYEAR scalar function
 - CLI applications 182
- DB2 copies
 - CLI/ODBC applications 257
- DB2 for Linux, UNIX, and Windows
 - automatic client reroute support 196
 - connections
 - application programming for high availability 209
 - high-availability support 202
- DB2 for z/OS
 - client configuration
 - automatic client reroute support 233
 - Sysplex workload balancing 233
 - high availability 231, 247
 - Sysplex workload balancing 231, 241
 - workload balancing 244
- DB2 Information Center
 - updating 287, 288
 - versions 286
- DB2_DIAGPATH variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
- DB2_ENABLE_LDAP variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
- DB2_FORCE_NLS_CACHE registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
- DB2_NO_FORK_CHECK registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
- DB2ACCOUNT registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
- DB2BIDI registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
- db2cli.ini file
 - details 12
- DB2CLIINIPATH variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17

- DB2CODEPAGE registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2DOMAINLIST variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2GRAPHICUNICODESERVER registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2LDAP_BASEDN variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2LDAP_CLIENT_PROVIDER registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2LDAP_KEEP_CONNECTION registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2LDAP_SEARCH_SCOPE variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2LDAPHOST variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2LOCALE registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2NOEXITLIST registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - db2oreg1.exe utility 18
 - DB2SORCVBUF variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2SOSNDBUF variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2TCP_CLIENT_RCVTIMEOUT registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DB2TERRITORY registry variable
 - IBM Data Server Driver for ODBC and CLI environment variable 17
 - DBCLOB data type
 - details 62
 - deferred arguments 75
 - deferred prepares 87
 - DEGREES scalar function
 - CLI applications 182
 - descriptor handles
 - details 139
 - descriptors
 - allocating 143
 - concise functions 147
 - consistency checks 143
 - copying
 - CLI applications 146
 - overview 146
 - freeing 143
 - overview 139
 - DiagLevel CLI/ODBC keyword 40
 - DiagPath CLI/ODBC keyword 41
 - DIFFERENCE scalar function
 - CLI applications 182
 - distinct types
 - CLI applications 70
 - Distributed Transaction Coordinator (DTC)
 - registering XA library by using db2oreg1.exe utility 18
 - distributed transactions
 - client support for XA 249
 - distributed units of work
 - CICS 168
 - Encina 168
 - overview 165
 - transaction managers
 - DB2 166
 - process-based 168
 - documentation
 - overview 283
 - PDF files 283
 - printed 283
 - terms and conditions of use 290
 - driver managers
 - DataDirect ODBC 53
 - Microsoft ODBC 53
 - overview 51
 - unixODBC 51
 - drivers
 - CLI 3, 7
 - ODBC 3, 7
 - DTC (Distributed Transaction Coordinator)
 - registering XA library by using db2oreg1.exe utility 18
- ## E
- embedded SQL applications
 - C/C++
 - combining CLI and embedded SQL 127
 - CLI
 - combining CLI and embedded SQL 127
 - enableACR IBM Data Server Driver configuration keyword
 - DB2 for Linux, UNIX, and Windows 233
 - DB2 for z/OS 233
 - enableACR IBM Data Server Driver configuration parameter
 - DB2 for Linux, UNIX, and Windows 196
 - enableAlternateGroupSeamlessACR IBM Data Server Driver configuration keyword
 - DB2 for Linux, UNIX, and Windows 233
 - DB2 for z/OS 233
 - enableAlternateGroupSeamlessACR IBM Data Server Driver configuration parameter
 - DB2 for Linux, UNIX, and Windows 196
 - enableAlternateServerListFirstConnect IBM Data Server Driver configuration parameter
 - DB2 for Linux, UNIX, and Windows 196
 - enableDirectXA parameter 250
 - enableSeamlessACR IBM Data Server Driver configuration keyword
 - DB2 for Linux, UNIX, and Windows 233
 - DB2 for z/OS 233
 - enableSeamlessACR IBM Data Server Driver configuration parameter
 - DB2 for Linux, UNIX, and Windows 196
 - enableWLB IBM Data Server Driver configuration keyword
 - DB2 for Linux, UNIX, and Windows 233
 - Encina environmental configuration 168
 - ESCAPE clauses
 - vendor 179
 - examples
 - distinct types
 - CLI applications 70
 - EXP scalar function
 - CLI applications 182

F

- fetches
 - LOB data in CLI 111
- file DSN
 - protocol used 34
- file input/output for LOB data in CLI 66
- FileDSN CLI/ODBC keyword 32
- FLOOR function
 - CLI applications 182
- freeing CLI handles
 - overview 129
- freeing statement resources in CLI 128

H

- handles
 - descriptors 139
 - freeing
 - methods 129
 - types 57
- help
 - SQL statements 286
- high availability
 - client application 193
 - configuring
 - clients 217
 - non-Java client support 194
- high-availability cluster support
 - Informix 216
- HOURL scalar function
 - CLI applications 182

I

- IBM data server clients
 - automatic client reroute
 - DB2 for Linux, UNIX, and Windows server 204
 - DB2 for z/OS server 241
 - Informix database server 221
- IBM Data Server Driver configuration keywords
 - acrRetryInterval
 - DB2 for Linux, UNIX, and Windows 196
 - DB2 for z/OS 233
 - alternateserverlist
 - DB2 for Linux, UNIX, and Windows 196
 - automatic client reroute 233
 - connectionLevelLoadBalancing
 - DB2 for z/OS 233
 - enableACR
 - DB2 for Linux, UNIX, and Windows 196, 233
 - DB2 for z/OS 233
 - enableAlternateGroupSeamlessACR
 - DB2 for Linux, UNIX, and Windows 196, 233
 - DB2 for z/OS 233
 - enableAlternateServerListFirstConnect
 - DB2 for Linux, UNIX, and Windows 196
 - enableSeamlessACR
 - DB2 for Linux, UNIX, and Windows 196, 233
 - DB2 for z/OS 233
 - enableWLB
 - DB2 for Linux, UNIX, and Windows 233
 - maxAcrRetries
 - DB2 for Linux, UNIX, and Windows 196
 - DB2 for z/OS 233
 - maxRefreshInterval
 - DB2 for Linux, UNIX, and Windows 233

IBM Data Server Driver configuration keywords *(continued)*

- maxTransportIdleTime
 - DB2 for z/OS 233
- maxTransports
 - DB2 for z/OS 233
- maxTransportWaitTime
 - DB2 for z/OS 233
- Sysplex workload balancing 233
- IBM Data Server Driver for ODBC and CLI
 - applications 35
 - CLI functions 35
 - CLI trace 38
 - configuring
 - environment variables 15
 - Microsoft DTC 19
 - Microsoft ODBC driver manager 19
 - procedure 11
 - registering ODBC data sources 23
 - connecting to databases 20
 - db2diag log files 38
 - db2support utility 38
 - db2trc utility 38
 - deploying with applications 49
 - environment variables 17
 - installing 8, 9, 11
 - LDAP support 37
 - license requirements 49
 - obtaining 8
 - ODBC functions 35
 - overview 7
 - problem determination 38
 - registering ODBC data sources 23
 - restrictions 37
 - security plug-ins 25
 - XA functions 36
- IBM data server drivers
 - automatic client reroute
 - DB2 for Linux, UNIX, and Windows server 204
 - DB2 for z/OS server 241
 - Informix database server 221
- IFNULL scalar function 182
- implementation parameter descriptor (IPD) 139
- implementation row descriptor (IRD) 139
- imports
 - data 114
- Informix
 - high availability 216, 220, 224
 - workload balancing 223
- INI file 12
- INSERT scalar function 182
- Instance CLI/ODBC keyword 32
- INSTANCE variable 17
- Interrupt CLI/ODBC keyword 33
- IPD (implementation parameter descriptor) 139
- IRD (implementation row descriptor) 139
- isolation levels
 - ODBC 3

J

- JDBC
 - static profiling 126
- JULIAN_DAY scalar function
 - details 182

K

- keysets 94
- KRBPlugin CLI/ODBC keyword 33
- KRBPLUGIN variable 17

L

- large objects (LOBs)
 - CLI applications 62, 66, 111
 - CLI file input and output 66
 - direct file input in CLI applications 66
 - direct file output in CLI applications 66
 - fetching
 - locators in CLI applications 111
 - locators
 - CLI applications 64
 - LongDataCompat CLI/ODBC keyword 67
 - ODBC applications 67
- LCASE (locale sensitive) scalar function
 - details 182
- LDAP
 - IBM Data Server Driver for ODBC and CLI 37
- LEFT scalar function
 - CLI applications 182
- LENGTH scalar function
 - CLI applications 182
- licenses
 - IBM Data Server Driver for ODBC and CLI 49
- link options
 - AIX
 - CLI applications 262
 - CLI routines 276
 - HP-UX
 - CLI applications 262
 - Linux
 - CLI applications 263
 - Solaris
 - CLI applications 264
 - Windows
 - CLI applications 268
- Linux
 - ODBC environment 251
- load utility
 - CLI applications 114
- LOCATE scalar function
 - CLI applications 182
- LOG scalar function 182
- LOG10 scalar function
 - CLI applications 182
- long data
 - CLI 68
 - retrieving data in pieces 82
 - sending data in pieces 82
- LongDataCompat CLI/ODBC configuration keyword
 - accessing LOB columns 67
- lowercase conversion scalar function 182
- LTRIM scalar function
 - CLI applications 182

M

- maxAcrRetries IBM Data Server Driver configuration parameter
 - DB2 for Linux, UNIX, and Windows 196
 - DB2 for z/OS 233

- maxRefreshInterval IBM Data Server Driver configuration
 - keyword 233
- maxTransportIdleTime IBM Data Server Driver configuration
 - keyword 233
- maxTransports IBM Data Server Driver configuration
 - keyword 233
- maxTransportWaitTime IBM Data Server Driver configuration
 - keyword 233
- metadata
 - characters 150
- Microsoft DTC
 - configuring IBM Data Server Driver for ODBC and CLI 19
- Microsoft ODBC driver manager
 - CLI comparison 3
 - configuring IBM Data Server Driver for ODBC and CLI 19
- MINUTE scalar function
 - CLI applications 182
- MOD function
 - CLI applications 182
- MONTH scalar function
 - CLI applications 182
- MONTHNAME scalar function
 - CLI applications 182
- multi-threaded applications
 - CLI applications 175, 176
- multisite updates
 - CLI applications 165

N

- non-Java clients
 - automatic client reroute
 - DB2 for Linux, UNIX, and Windows server 204
- notices 293
- NotifyLevel CLI/ODBC keyword 40
- NOW scalar function 182
- null-terminated strings in CLI applications 60

O

- ODBC
 - CLI 1, 3
 - driver managers
 - unixODBC 51, 253
 - drivers
 - overview 7
 - IBM Data Server Driver for ODBC and CLI
 - CLI functions 35
 - configuring 11, 15, 19
 - connecting to databases 20
 - DB2 registry variables 17
 - deploying with applications 49
 - installing 8, 9, 11
 - LDAP support 37
 - license requirements 49
 - obtaining 8
 - ODBC functions 35
 - overview 7
 - problem determination 38
 - registering ODBC data sources 23
 - restrictions 37
 - running database applications 35
 - XA functions 36

- ODBC (*continued*)
 - IBM DB2 Driver for ODBC and CLI
 - environment variables 17
 - isolation levels 3
 - registering ODBC data sources 23
 - registering XA library with DTC 18
 - setting up environment (Linux and UNIX) 251
 - vendor escape clauses 179
- offsets
 - binding columns 109
 - changing parameter bindings 81

P

- packages
 - bind option limitations 260
 - names
 - binding 258
- parameter markers
 - binding
 - changing 81
 - CLI applications 75, 78
 - column-wise array input in CLI 79
 - row-wise array input in CLI 79
- parameter status arrays 80
- parameters
 - CLI applications 80
- parsing
 - explicit
 - CLI applications 117
 - implicit
 - CLI applications 117
- pattern values 150
- percent signs
 - catalog functions 150
 - LIKE predicates 150
- performance
 - CLI array input chaining 159
- PI scalar function 182
- POWER scalar function
 - details 182
- prepared SQL statements
 - CLI applications
 - creating 86
- problem determination
 - information available 290
 - tutorials 290
- process-based transaction manager 168
- Protocol CLI/ODBC configuration keyword 34
- PROTOCOL variable 17
- PWDPlugin CLI/ODBC keyword 34
- PWDPLUGIN variable 17

Q

- QUARTER scalar function
 - CLI applications 182
- queries
 - system catalog information 149

R

- RADIANS scalar function
 - CLI applications 182
- RAND scalar function
 - CLI applications 182

- reentrance 175
- REPEAT scalar function
 - overview 182
- REPLACE scalar function
 - overview 182
- result sets
 - CLI
 - specifying rowset returned from 100
 - terminology 94
- retrieving data
 - arrays
 - column-wise binding 108
 - row-wise binding 108
 - CLI
 - arrays 106
 - bookmarks 104, 105
 - pieces 110
 - query results 97
 - row sets 96
 - scrollable cursors 102
 - XML
 - CLI applications 112
- RIGHT scalar function
 - vendor escape clauses 182
- rollbacks
 - transactions 83
- ROUND scalar function
 - vendor escape clauses 182
- row sets
 - CLI functions
 - retrieval examples 96
 - specifying 100
 - terminology in CLI applications 94
- row-wise binding 106, 108
- RTRIM scalar function
 - vendor escape clauses 182

S

- SaveFile CLI/ODBC keyword 34
- search conditions
 - input to catalog functions 150
- SECOND scalar function
 - CLI applications 182
- SECONDS_SINCE_MIDNIGHT scalar function 182
- security
 - plug-ins
 - IBM Data Server Driver for ODBC and CLI 25
- serialization
 - explicit
 - CLI applications 112
 - implicit
 - CLI applications 71, 112
- SIGN scalar function
 - overview 182
- SIN scalar function
 - overview 182
- SOUNDEX scalar function
 - CLI applications 182
- SPACE scalar function
 - CLI applications 182
- SQL
 - parameter markers 75
- SQL Access Group 1
- SQL statements
 - CLI applications 74
 - freeing resources in CLI 128

SQL statements (*continued*)

- help
 - displaying 286
- SQL_ATTR_
 - CONNECTION_POOLING environment attribute 41
 - CONNECTTYPE 166
 - ConnectType CLI/ODBC configuration keyword 165
 - environment attribute 41
 - CP_MATCH environment attribute 41
 - DIAGLEVEL environment attribute 41
 - DIAGPATH environment attribute 41
 - INFO_ACCTSTR
 - environment attribute 41
 - INFO_APPLNAME
 - environment attribute 41
 - INFO_USERID
 - environment attribute 41
 - INFO_WRKSTNNAME
 - environment attribute 41
 - LONGDATA_COMPAT 67
 - MAXCONN
 - environment attribute 41
 - NOTIFYLEVEL environment attribute 41
 - ODBC_VERSION environment attribute 41
 - OUTPUT_NTS 41
 - PROCESSCTRL
 - environment attribute 41
 - RESET_CONNECTION
 - environment attribute 41
 - SYNC_POINT
 - environment attribute 41
 - TRACE
 - environment attribute 41
 - TRACENOHEADER environment attribute 41
 - TRUSTED_CONTEXT_PASSWORD
 - switching users on a trusted connection through CLI 135
 - TRUSTED_CONTEXT_USERID
 - switching users on a trusted connection through CLI 135
 - USE_2BYTES_OCTET_LENGTH environment attribute 41
 - USE_LIGHT_INPUT_SQLDA environment attribute 41
 - USE_LIGHT_OUTPUT_SQLDA environment attribute 41
 - USE_TRUSTED_CONTEXT
 - creating trusted connection through CLI 134
 - USER_REGISTRY_NAME
 - environment attribute 41
- SQL_CONCURRENT_TRANS value of
 - SQL_ATTR_CONNECTTYPE environment attribute 166
- SQL_COORDINATED_TRANS value of
 - SQL_ATTR_CONNECTTYPE environment attribute 166
- SQL_NTS 60
- SQL_ONEPHASE 166
- SQL_TWOPHASE 166
- SQLBindCol CLI function
 - position in typical order of function calls 73
- SQLBindParameter CLI function
 - parameter marker binding 75
- SQLBrowseConnect CLI function
 - Unicode version 162
- SQLBrowseConnectW CLI function 162
- SQLBulkOperations CLI function
 - deleting bulk data 120
 - inserting bulk data 113
 - retrieving bulk data 105
 - updating bulk data 119
- SQLColAttribute CLI function
 - Unicode version 162
- SQLColAttributes CLI function
 - Unicode version 162
- SQLColAttributesW CLI function 162
- SQLColAttributeW CLI function 162
- SQLColumnPrivileges CLI function
 - Unicode version 162
- SQLColumnPrivilegesW CLI function 162
- SQLColumns CLI function
 - Unicode version 162
- SQLColumnsW CLI function 162
- SQLConnect CLI function
 - Unicode version 162
- SQLConnectW CLI function 162
- SQLCreateDbW CLI function 162
- SQLDataSources CLI function
 - Unicode version 162
- SQLDataSourcesW CLI function 162
- SQLDescribeCol CLI function
 - position in typical order of function calls 73
 - Unicode version 162
- SQLDescribeColW CLI function 162
- SQLDriverConnect CLI function
 - details 25
 - Unicode version 162
- SQLDriverConnectW CLI function 162
- SQLDropDbW CLI function 162
- SQLEndTran CLI function
 - need for 85
- SQLError deprecated CLI function
 - Unicode version 162
- SQLErrorW CLI function 162
- SQLExecDirect CLI function
 - position in typical order of function calls 73
 - Unicode version 162
- SQLExecDirectW CLI function 162
- SQLExecute CLI function
 - position in typical order of function calls 73
- SQLExtendedPrepare CLI function
 - Unicode version 162
- SQLExtendedPrepareW CLI function 162
- SQLExtendedProcedureColumns
 - Unicode version 162
- SQLExtendedProcedureColumnsW CLI function 162
- SQLExtendedProcedures
 - Unicode version 162
- SQLExtendedProceduresW CLI function 162
- SQLFetch CLI function
 - position in typical order of function calls 73
- SQLForeignKeys CLI function
 - Unicode version 162
- SQLForeignKeysW CLI function 162
- SQLGetConnectAttr CLI function
 - Unicode version 162
- SQLGetConnectAttrW CLI function 162
- SQLGetConnectOption deprecated CLI function
 - Unicode version 162
- SQLGetConnectOptionW CLI function 162
- SQLGetCursorName CLI function
 - Unicode version 162
- SQLGetCursorNameW CLI function 162
- SQLGetData CLI function
 - position in typical order of function calls 73
- SQLGetDescField CLI function
 - Unicode version 162
- SQLGetDescFieldW CLI function 162

- SQLGetDescRec CLI function
 - Unicode version 162
- SQLGetDescRecW CLI function 162
- SQLGetDiagField CLI function
 - Unicode version 162
- SQLGetDiagFieldW CLI function 162
- SQLGetDiagRec CLI function
 - Unicode version 162
- SQLGetDiagRecW CLI function 162
- SQLGetInfo CLI function
 - Unicode version 162
- SQLGetInfoW CLI function 162
- SQLGetPosition CLI function
 - Unicode version 162
- SQLGetStmtAttr CLI function
 - Unicode version 162
- SQLGetStmtAttrW CLI function 162
- SQLNativeSql CLI function
 - Unicode version 162
- SQLNativeSqlW CLI function 162
- SQLNumResultCols CLI function
 - position in typical order of function calls 73
- SQLPrepare CLI function
 - position in typical order of function calls 73
 - Unicode version 162
- SQLPrepareW CLI function 162
- SQLPrimaryKeys CLI function
 - Unicode version 162
- SQLPrimaryKeysW CLI function 162
- SQLProcedureColumns CLI function
 - Unicode version 162
- SQLProcedureColumnsW CLI function 162
- SQLProcedures CLI function
 - Unicode version 162
- SQLProceduresW CLI function 162
- SQLReloadConfig CLI function
 - Unicode version 162
- SQLReloadConfigW CLI function 162
- SQLRowCount CLI function
 - position in typical order of function calls 73
- SQLSetConnectAttr CLI function
 - Unicode version 162
- SQLSetConnectAttrW CLI function 162
- SQLSetConnectOption deprecated CLI function
 - Unicode version 162
- SQLSetConnectOptionW CLI function 162
- SQLSetCursorName CLI function
 - Unicode version 162
- SQLSetCursorNameW CLI function 162
- SQLSetDescField CLI function
 - Unicode version 162
- SQLSetDescFieldW CLI function 162
- SQLSetStmtAttr CLI function
 - Unicode version 162
- SQLSetStmtAttrW CLI function 162
- SQLSpecialColumns CLI function
 - Unicode version 162
- SQLSpecialColumnsW CLI function 162
- SQLStatistics CLI function
 - Unicode version 162
- SQLStatisticsW CLI function 162
- SQLTablePrivileges CLI function
 - Unicode version 162
- SQLTablePrivilegesW CLI function 162
- SQLTables CLI function
 - Unicode version 162
- SQLTablesW CLI function 162

- SQRT scalar function
 - CLI applications 182
- statement handles
 - allocating 74
- stored procedures
 - calling
 - CLI applications 120
 - ODBC escape clause 179
- strings
 - input arguments 60
 - length in CLI applications 60
- SUBSTRING scalar function
 - CLI applications 182
- Sysplex workload balancing
 - DB2 for z/OS 231
 - direct connections to DB2 for z/OS 241
 - example of enabling 239
- Sysplex workload balancing for DB2 for z/OS
 - client configuration 233
- system catalogs
 - querying 149

T

- TAN scalar function
 - CLI applications 182
- termination
 - CLI applications 131
 - task 56
- terms and conditions
 - publications 290
- threads
 - multiple
 - CLI applications 175
- TIMESTAMPDIFF scalar function
 - CLI applications 182
- transaction managers
 - CLI applications
 - configuring 166
 - programming considerations 168
- transaction-level load balancing
 - client application 193
- transactions
 - commits 83
 - ending in CLI 85
 - rollbacks 83
- troubleshooting
 - online information 290
 - tutorials 290
- TRUNC scalar function
 - CLI applications 182
- TRUNCATE scalar function
 - CLI applications 182
- truncation
 - output strings 60
- trust relationships
 - DB2 Connect 133
- trusted connections
 - CLI/ODBC 134
 - DB2 Connect 133
 - switching users through CLI/ODBC 135
- trusted contexts
 - CLI/ODBC support 134
 - DB2 Connect support 133
- tutorials
 - list 290
 - problem determination 290

tutorials (*continued*)
 pureXML 290
 troubleshooting 290
two-phase commit
 CLI 165

U

UCASE scalar function
 CLI applications 182
UDTs
 CLI applications 69
 distinct types
 CLI applications 70
underscores
 catalog functions 150
 LIKE predicates 150
Unicode UCS-2 encoding
 CLI
 applications 161
 functions 162
 ODBC driver managers 163
units of work
 distributed 83
UNIX
 ODBC environment setup 251
unixODBC driver manager
 build scripts 253
 configurations 253
 setting up 51
updates
 bulk data with bookmarks in CLI 119
 data
 CLI applications 118
 DB2 Information Center 287, 288
USER scalar function 182

V

vendor escape clauses 179

W

WEEK scalar function
 CLI applications 182
WEEK_ISO scalar function
 CLI applications 182
Windows
 CLI environment setup 255
workload balancing
 connections to DB2 for Linux, UNIX, and Windows 206
 connections to DB2 for z/OS 244
 connections to Informix 223
 DB2 for Linux, UNIX, and Windows
 client configuration 201

X

X/Open Company 1
X/Open SQL CLI 1
XA
 client-side support 249
 IBM Data Server Driver for ODBC and CLI 36
 registering XA library with DTC 18
 trusted connections 133

XA support
 enabling for clients 250
XML
 parsing
 CLI applications 117
 serialization
 CLI applications 71, 112
XML data
 CLI applications
 inserting 117
 overview 71
 retrieval 112
 updating 117
XML data retrieval
 CLI applications 112
XML data type
 CLI applications 71

Y

YEAR scalar function
 CLI applications 182



Printed in USA

SC27-3866-01



Spine information:

IBM DB2 10.1 for Linux, UNIX, and Windows

Call Level Interface Guide and Reference Volume 1

